

Scalable Minimal-Change Learning for Controllable Image Editing

Shuo Chen¹, Fengming Huang¹, Yu Yao^{1✉}, Mingming Gong², Tongliang Liu¹

¹Sydney AI Centre, The University of Sydney

²University of Melbourne

yu.yao@sydney.edu.au

Abstract

Image editing aims to modify specific attributes of an image while preserving all other aspects. In practice, however, applying even simple editing instructions to existing methods often leads to unintended changes. We identify the central issue: a fundamental causal principle of *minimal change*, which requires that an intervention alter only the intended attributes in the output while leaving all others invariant, has not been explicitly incorporated as an optimization objective. To encourage minimal change, prior work rooted in causal representation learning typically imposes an L_1 regularizer on the latent difference between pre- and post-edit representations. However, this strategy does not scale to modern image editing models: L_1 regularization fails to induce true sparsity and instead tends to shrink all latent differences uniformly; sparsity in the latent space does not translate to localized change in the output of nonlinear deep image generation models; and the multi-domain or counterfactual supervision it requires is generally unavailable. To effectively leverage the minimal change principle for controllable image editing, we cast it as a reinforcement learning objective. Rather than constraining latent representations, we design rewards that directly capture minimal change over intervention outcomes and optimize them explicitly during training. To produce these rewards reliably, we further introduce an agentic reward model that leverages the chain-of-thought reasoning ability of vision-language models without requiring human annotations. The idea is that instead of emitting a single, unreliable scalar score, the model reasons explicitly about two complementary failure modes, unimplemented changes and unintended changes, and produces this supervision automatically. Across four benchmarks, ARRO improves average EditScore from 5.21 to 5.88. A region-aware evaluation on 600 examples further shows an 8.4% reduction in off-target pixel change relative to the base editor. Controlled reward and SFT comparisons, blinded human evaluations, and transfer to OmniGen2 provide complementary evidence for the reward design and its applicability beyond one backbone. **Code:** <https://github.com/Showwwwwww/ARRO>.

1 Introduction

Image editing aims to modify specific attributes of an image while preserving all other aspects. Recent instruction-based editors [Brooks et al., 2023, Zhang et al., 2023, Sheynin et al., 2023, Black Forest Labs et al., 2025, Wu et al., 2025a,b, Deng et al., 2025, Zhang et al., 2025] have made impressive progress on visual quality and instruction following. Yet even simple instructions still trigger widespread, unintended changes: “add glasses” subtly alters the subject’s hair and skin; “change the car’s color” redraws the background; “remove the sign” shifts the lighting of the entire scene. This is not a cosmetic issue but a fundamental one: it determines whether a generative model can be trusted as a controllable editing tool, rather than a high-fidelity sampler that happens to follow instructions most of the time.

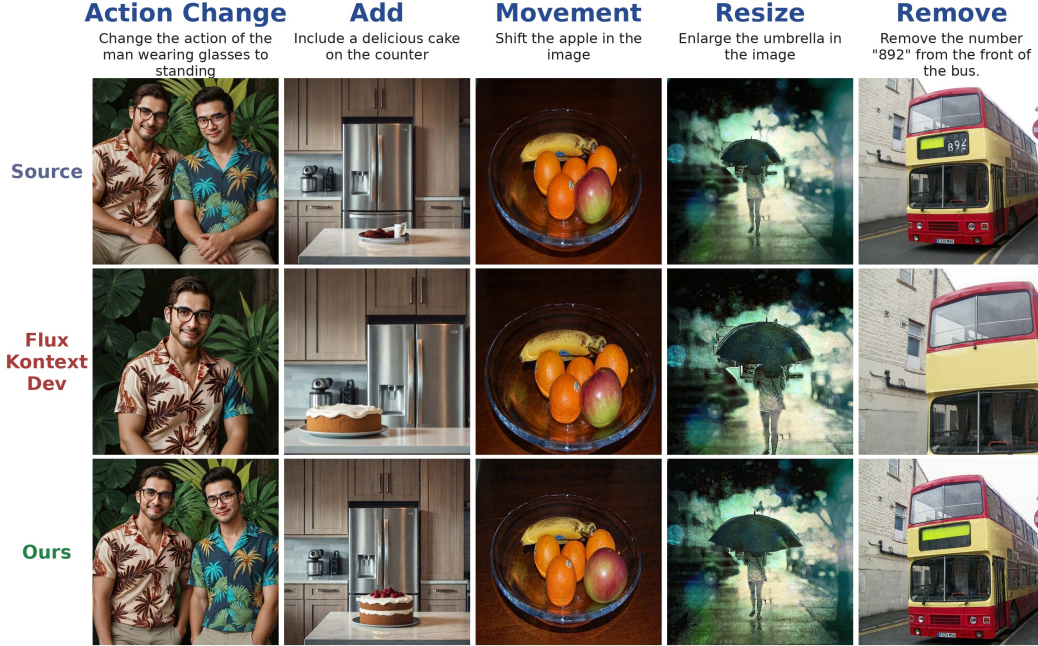


Figure 1: Illustrative edits by FLUX.1 Kontext-dev and ARRO. ARRO better preserves the companion, fruit arrangement, umbrella scene, and bus surroundings while carrying out the requested edits. The ambiguous “Include a delicious cake” example remains imperfect: ARRO adds a cake but replaces the original dish. It is retained as a visible failure case rather than evidence of perfect preservation.

We identify a central yet under-explored issue behind this gap: the fundamental causal principle of *minimal change*, which requires that an intervention alter only the intended attributes in the output while leaving all others invariant, has not been explicitly incorporated as an optimization objective in modern image editing models.

Minimal change is widely applied in causal representation learning [Schölkopf et al., 2021, Peters et al., 2017, Perry et al., 2022, Lopez et al., 2023]. Existing methods (e.g., [Yang et al., 2021, Shen et al., 2022, Komanduri et al., 2022]) operationalize it by imposing an L_1 regularizer on the latent difference between pre- and post-intervention representations, encouraging interventions on a single latent factor to leave the remaining factors unchanged. This strategy is difficult to adapt effectively to modern image editing models, due to three obstacles. **(O1)** L_1 does not reliably induce sparsity: it tends to shrink all latent coordinates uniformly toward small values rather than driving most of them to exactly zero [Shabrina et al., 2025], so latent edits remain dense, just quieter. **(O2)** Latent sparsity does not translate to output locality: modern decoders are large, deeply nonlinear neural networks with a latent-to-image Jacobian that is far from axis-aligned, so even a sparse change in the latent space can propagate into broad, unintended modifications across the output image [Shen et al., 2020, Karras et al., 2020]. **(O3)** The required supervision does not scale: causal representation learning methods rely on multi-domain datasets with attribute labels or paired counterfactual examples, which are expensive or generally unavailable.

We address all three obstacles by reformulating minimal change as a reinforcement learning (RL) objective on intervention outcomes rather than latents. Designing rewards that directly capture minimal change on the output dissolves **(O1)** and **(O2)** simultaneously: there is no L_1 approximation to optimize, and the decoder’s nonlinearity is no longer a problem, because the reward is computed directly on the output rather than on the latent that produces it. Concretely, the supervised dataset contains only source-instruction pairs (x_{src}, c) ; during RL, the current policy samples candidate edits x_i , which are then assigned minimal-change rewards r_i according to how well they implement the instruction while leaving everything else intact.

To address **(O3)**, a minimal-change reward has to be reliably designed without human annotation. A natural choice would be to ask a vision-language model (VLM) for a single editing quality score, as in recent reward-modeling work [Gong et al., 2025, Luo et al., 2025]. However, directly prompting a VLM for a numerical score is unreliable. First, the score is poorly calibrated: scalar VLM ratings are noisy and high-variance, with thresholds that shift across images, so the same numerical score can mean different things on different inputs [Li et al., 2026]. Second, the score is uninformative about the type of failure: a minimal-change reward must capture two distinct aspects, whether the instructed edit was carried out and whether everything else was left alone, but a single number collapses both into one signal, so the policy cannot tell which aspect it is failing on.

To make the minimal-change reward more reliable, we design an agentic reward model: rather than asking the VLM for a numerical score, we leverage its chain-of-thought reasoning ability to perform a structured audit of the edit. We decompose the minimal-change reward into two complementary components: *unimplemented changes*, where the instructed edit is missing or incomplete, and *unintended changes*, where modifications appear outside the instructed scope. For each component, the VLM reasons over the editing triplet of source image, instruction, and edited output, and produces a list of identified issues. To stabilize the agentic reward model, we further introduce a *rubric refinement* loop that enforces consistency within each GDPO group (the set of candidate edits sampled for a given source-instruction pair). An LLM first merges the candidates’ audit lists into a single deduplicated rubric. The VLM then performs a second-pass evaluation in which, for each candidate, it selects the applicable items from this shared rubric, and the two reward signals are computed from the selected items.

On top of FLUX.1 Kontext, our method lifts average EditScore from 5.21 to 5.88 across four benchmarks (MinEval, MagicBrush, AnyBench, and Emu-Edit). A masked pixel-difference evaluation on 600 examples measures 8.4% less off-target change than the base editor. Under fixed GDPO training, replacing ARRO’s rewards with EditScore Prompt-Following and Consistency rewards yields 3.78 average EditScore, while ARRO achieves 5.88. On MinEval, ARRO-GDPO also exceeds reward-guided Top-10 SFT (5.669 vs. 4.629 EditScore). Blinded human comparisons and transfer to OmniGen2 provide further evidence. These results support supervising the *effect* of an intervention.

Contributions. (1) We diagnose three reasons why latent-space sparsity fails to deliver minimal change in modern generative editors ineffective L_1 regularization, decoder nonlinearity, and the lack of scalable supervision and recast minimal change as a reinforcement learning objective on intervention outcomes, optimized end-to-end over a flow-matching editor. (2) We design an agentic reward model that, unlike scalar VLM judges, reasons over the editing triplet to produce decomposed efficacy and locality rewards from raw images alone counting unimplemented and unintended changes, respectively stabilized by a group-level rubric refinement step that anchors all candidates to a shared, deduplicated set of canonical issues. (3) We developed MinEval, a dataset of automatically generated edit prompts and source images covering atomic changes across six edit types, with disjoint training and held-out evaluation splits. MinEval is designed to facilitate research into controllable image editing. (4) On top of FLUX.1 Kontext, ARRO improves average EditScore from 5.21 to 5.88 across four benchmarks and reduces masked off-target pixel change by 8.4% on a 600-example evaluation. Reward-pipeline, EditScore-reward, and Top-10 SFT controls, human evaluations, and OmniGen2 transfer probe the source and breadth of these gains (Sec. 4; App. C.1).

2 Related Work

Instruction-based Image Editing. Following Brooks et al. [2023], recent models like FLUX.1 Kontext [Black Forest Labs et al., 2025] treat locality as an emergent property of pretraining. However, unintended background drift persists in current benchmarks [Qian et al., 2025, Jiang et al., 2026]. Post-training approaches address this via preference optimization [Wallace et al., 2024, Zhao et al., 2025] or online RL [Liu et al., 2025, Li et al., 2026] using learned reward models [Luo et al., 2025, Gong et al., 2025]. We depart from this line in two ways: (1) we argue human preference rewards often conflate aesthetics with adherence rather than penalizing non-minimal changes, and (2) our agentic VLM reward is zero-shot, bypassing the need for expensive, learned reward-model training.

Flow Matching and Sampling Efficiency. Research on generative flow matching has improved few-step synthesis through model-aligned training couplings [Lin et al., 2025b] and sub-mode

conditioning for diverse one-step generation [Lin et al., 2026]. In a distinct, discrete-diffusion setting, parallel-in-time sampling accelerates the sequential generation process [Yao et al., 2026]. These works address transport, diversity, and sampling cost; our objective concerns the scope of changes made by an instruction-conditioned editor.

The Minimal Change Principle. The principle that interventions should perturb only targeted mechanisms originates in causal inference [Pearl, 2009, Schölkopf et al., 2021]. In generative modeling, this is often operationalized through structured latent spaces [Yang et al., 2021, Komanduri et al., 2024] or training-free latent constraints [Kamenetsky et al., 2025, Shabrina et al., 2025]. While we share the conceptual goal of sparse interventions, we argue that enforcing minimal change on a latent passed through a non-linear decoder is a brittle proxy. Instead, we optimize for minimal change directly in the output space via visual auditing.

Task-specific representations and metrics. EditCLIP [Wang et al., 2025] learns representations of source-to-edit transformations for exemplar-based editing and evaluation; NearID [Cvejic et al., 2026] uses matched-background distractors to learn identity representations; StyleID [Yun et al., 2026] learns a facial-identity metric from perceptual judgments across stylizations. These methods use supervision tailored to a predefined representation or metric. ARRO instead constructs instruction-specific, decomposed reward signals from candidate edits without per-instruction human labels; the resulting issue annotations can also support offline candidate selection. We do not claim that these representation-learning tasks are interchangeable with online editing rewards.

Safety and Asset Protection. Adjacent work examines visual-instruction jailbreaks of image-to-video models [Zheng et al., 2026], conflict-aware safety guidance for text-to-image diffusion [Xiang et al., 2026], protection of 3D Gaussian splatting assets from instruction-driven editing [Hong et al., 2025], and defenses against image-to-video generation and fine-tuning-based video customization [Huang et al., 2026]. These studies address misuse and unauthorized modification; our focus is preserving untargeted content when carrying out an authorized image edit.

Agentic VLM Rewards. Standard VLM-based rewards typically provide scalar scores [Rocamonde et al., 2023, Gong et al., 2025]. Recent work has introduced structured reasoning or binary checklists to improve interpretability [Li et al., 2026, Wu et al., 2025c]. Our approach sits at the intersection of agentic VLMs [Wu et al., 2025d] and RL; rather than a global similarity score, our reward is derived from a structured audit of “unimplemented” versus “unintended” changes. This provides a granular, per-region optimization signal that makes locality a directly optimizable property.

Due to the limited space, the extended related work is provided in Appendix A.

3 Method

We cast *minimal change* as a reinforcement learning objective in the output space of a generative editor: given a source image x_{src} and an editing instruction c , the policy π_θ should realize the instructed effect while leaving every other aspect of the output unchanged. This re-frames controllable image editing away from representation-level sparsity, which constrains *how* the model encodes factors, and toward outcome-level optimization, which constrains *what the output looks like* after the intervention. An outcome-level reward must therefore observe two complementary failure modes at once: *unimplemented changes*, where the instructed edit is missing or incomplete, and *unintended changes*, where modifications appear outside the instructed scope.

Figure 2 illustrates the pipeline. Our method ARRO realizes this view in two stages: an agentic VLM audits each candidate edit and decomposes it into unimplemented and unintended lists (Sec. 3.2); a rubric refinement step then makes these audits group-consistent and yields two scalar rewards $r_i^{(1)}, r_i^{(2)}$ commensurable across the group (Sec. 3.3). The two channels are each group-normalized, their advantages combined, and the result drives a flow-based GDPO update (Sec. 3.4) on top of FLUX.1 Kontext [Black Forest Labs et al., 2025].

Notation. Source $x_{\text{src}} \in \mathcal{X}$, instruction c , dataset $\mathcal{D} = \{(x_{\text{src}}^{(k)}, c^{(k)})\}$, flow-based policy $\pi_\theta(x | x_{\text{src}}, c)$. The reward model invokes a VLM Φ_V and a text-only LLM Φ_L through three fixed prompts: audit P_{audit} , aggregation P_{agg} , verification P_{verify} (full templates in App. F).

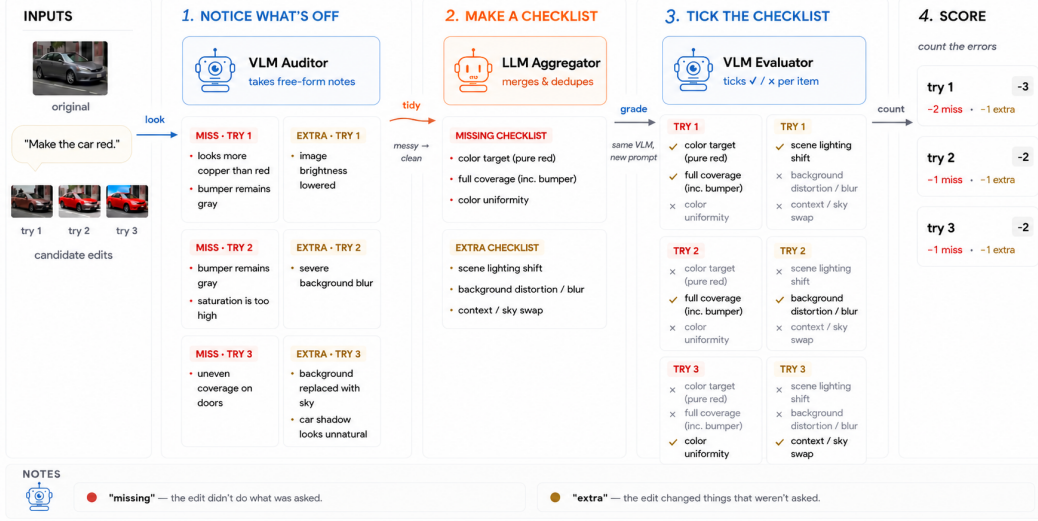


Figure 2: Overview of our agentic reward pipeline. For each editing triplet (x_{src}, c, x_i) , the vision-language model Φ_V performs a chain-of-thought audit, producing two raw lists of unimplemented and unintended changes. An LLM Φ_L aggregates these lists across the GDPO group into deduplicated rubrics, and Φ_V re-evaluates each candidate against the shared rubrics. The resulting two scalar rewards $(r_i^{(1)}, r_i^{(2)})$ are each group-normalized into per-channel advantages $(A_i^{(1)}, A_i^{(2)})$, which are then combined into the final advantage that drives the flow-based GDPO update.

3.1 Preliminaries

Rectified-flow editing model. FLUX.1 Kontext is a rectified-flow transformer [Liu et al., 2022, Black Forest Labs et al., 2025] that conditions on a reference image by concatenating its VAE latent z_{src} with the noisy latent z_t along the sequence axis. With $z_0 \sim p_{\text{data}}$ (the target-image latent), $\epsilon \sim \mathcal{N}(0, I)$, and $t \sim \mathcal{U}[0, 1]$, the velocity field v_θ is trained against the rectified-flow objective

$$\mathcal{L}_{\text{RF}}(\theta) = \mathbb{E}_{t, z_0, \epsilon} \left[\|v_\theta([z_t; z_{\text{src}}], t, c) - (\epsilon - z_0)\|_2^2 \right], \quad z_t = (1 - t)z_0 + t\epsilon. \quad (1)$$

We perform RL on top of a pretrained editor; the flow-matching loss is not used during fine-tuning.

GDPO. GDPO [Liu et al., 2026] adapts group-relative preference optimization to the multi-reward setting. For each input pair $(x_{\text{src}}, c) \sim \mathcal{D}$, it samples a group $\mathcal{G} = \{x_1, \dots, x_N\}$ from the current policy $\pi_{\theta_{\text{old}}}$ and assigns each candidate n scalar rewards $\{r_i^{(k)}\}_{k=1}^n$, with one per reward channel. Rather than collapsing the channels into a single scalar before normalization, GDPO z -scores each channel separately within $\mathcal{G}$,

$$A_i^{(k)} = \frac{r_i^{(k)} - \mu_{\mathcal{G}}^{(k)}}{\sigma_{\mathcal{G}}^{(k)} + \varepsilon_0}, \quad k = 1, \dots, n, \quad (2)$$

forms a (weighted) sum $A_i^{\text{sum}} = \sum_k w_k A_i^{(k)}$, and re-normalizes A_i^{sum} across the batch to keep the advantage scale stable as n grows. The policy is then updated by a per-step PPO-clip objective along an SDE discretization of the rectified flow [Li et al., 2025]. Full equations and the MixGRPO sampler are deferred to App. B.

3.2 Agentic Reward Model via Chain-of-Thought Auditing

We deploy the vision-language model Φ_V as an autonomous *causal observer*. Rather than asking the VLM Φ_V to regress a scalar score, which is unreliable for the calibration, variance, and conflation reasons discussed in Sec. 1, we prompt it to perform a multi-step audit of the editing triplet (x_{src}, c, x_i) for each candidate $x_i \in \mathcal{G}$.

Per-candidate audit. For each candidate x_i , the audit pass invokes the VLM Φ_V with the audit prompt P_{audit} instantiated on the editing triplet:

$$(\tilde{\mathcal{U}}_i, \tilde{\mathcal{O}}_i) = \Phi_V(P_{\text{audit}}(x_{\text{src}}, c, x_i)). \quad (3)$$

Audit prompt P_{audit} (a summary)

Inputs: source image x_{src} , edited image x_i , instruction c .

Step 1. Parse the instruction c : identify what should change and what should remain.

Step 2. Compare x_{src} and x_i side by side; note all visible differences.

Step 3. Emit two open-ended natural-language lists:

- the raw unimplemented list $\tilde{\mathcal{U}}_i$, enumerating aspects of c not realized in x_i ;
- the raw unintended list $\tilde{\mathcal{O}}_i$, enumerating visible changes outside the scope of c .

Output: JSON object containing $\tilde{\mathcal{U}}_i$ and $\tilde{\mathcal{O}}_i$.

The two outputs are ordered lists $\tilde{\mathcal{U}}_i = (u_{i,1}, u_{i,2}, \dots)$ and $\tilde{\mathcal{O}}_i = (e_{i,1}, e_{i,2}, \dots)$, where each entry $u_{i,j}$ or $e_{i,j}$ is a short natural-language description. Crucially, Φ_V never outputs a number on a continuous scale; it only identifies and describes changes, both tasks modern VLMs handle reliably. However, raw counts $|\tilde{\mathcal{U}}_i|$ and $|\tilde{\mathcal{O}}_i|$ are unreliable across candidates; We address this in Sec. 3.3.

3.3 Rubric-Refined Reward

Counting raw audit lists directly is unreliable: free-form VLM enumerations duplicate the same artifact under different wordings, fragment a single change into several entries, and drift in granularity across candidates, so $|\tilde{\mathcal{U}}_i|$ and $|\tilde{\mathcal{O}}_i|$ are not commensurable across $\mathcal{G}$. To fix this, we introduce the language model Φ_L as a *rubric aggregator* operating within each GDPO group.

Aggregation. The aggregator is invoked once per group. Its prompt P_{agg} receives both raw issue collections and returns one rubric for each failure mode:

$$(\mathcal{R}_{\mathcal{G}}^{\mathcal{U}}, \mathcal{R}_{\mathcal{G}}^{\mathcal{O}}) = \Phi_L(P_{\text{agg}}(c, \text{concat}_{i=1}^N \tilde{\mathcal{U}}_i, \text{concat}_{i=1}^N \tilde{\mathcal{O}}_i)). \quad (4)$$

Aggregation prompt P_{agg} (a summary)

Inputs: instruction c and both collections of raw audit lists across the group: $\text{concat}_{i=1}^N \tilde{\mathcal{U}}_i$ and $\text{concat}_{i=1}^N \tilde{\mathcal{O}}_i$.

Step 1 (instruction filter). Filter changes requested or directly implied by c out of the unintended collection; retain the unimplemented collection.

Step 2. Consolidate the entries: merge entries describing the same underlying issue under different wordings, split entries that conflate distinct issues, normalize the granularity of all entries to a single atomic visually-checkable change, and drop vague, speculative, or non-visual entries.

Step 3. Produce two separate deduplicated rubrics of canonical issues, written in neutral form so that any candidate can be checked against each entry.

Output: both $\mathcal{R}_{\mathcal{G}}^{\mathcal{U}}$ and $\mathcal{R}_{\mathcal{G}}^{\mathcal{O}}$ in one response.

The output unimplemented rubric $\mathcal{R}_{\mathcal{G}}^{\mathcal{U}} = \{u_1^*, \dots, u_{M_{\mathcal{U}}}^*\}$ and unintended rubric $\mathcal{R}_{\mathcal{G}}^{\mathcal{O}} = \{e_1^*, \dots, e_{M_{\mathcal{O}}}^*\}$ are deduplicated lists whose entries we call *canonical issues*. Together, the two rubrics form a per-group, group-consistent vocabulary against which every candidate in $\mathcal{G}$ can be evaluated on equal footing.

Rubric-anchored verification. A second VLM pass re-evaluates each candidate against both group rubrics in one verification call:

$$(\mathcal{U}_i, \mathcal{O}_i) = \Phi_V(P_{\text{verify}}(x_{\text{src}}, c, x_i, \mathcal{R}_{\mathcal{G}}^{\mathcal{U}}, \mathcal{R}_{\mathcal{G}}^{\mathcal{O}})). \quad (5)$$

Verification prompt P_{verify} (a summary)

Inputs: source image x_{src} , edited image x_i , instruction c , and both rubrics $\mathcal{R}_{\mathcal{G}}^{\mathcal{U}}$ and $\mathcal{R}_{\mathcal{G}}^{\mathcal{O}}$.

Step 1. Parse the instruction c (identical to Step 1 of P_{audit}).

Step 2. Compare x_{src} and x_i side by side (identical to Step 2 of P_{audit}).

Step 3. For each canonical issue in each rubric, return a Boolean verdict on whether that issue is visibly present in x_i . No new issues are admitted; the wording of rubric items is not modified.

Output: both verified subsets $\mathcal{U}_i \subseteq \mathcal{R}_{\mathcal{G}}^{\mathcal{U}}$ and $\mathcal{O}_i \subseteq \mathcal{R}_{\mathcal{G}}^{\mathcal{O}}$ in one response.

The verification prompt P_{verify} shares its first two reasoning steps with the audit prompt P_{audit} , that is, parsing the instruction c and comparing x_{src} with x_i , but its third step replaces open-ended generation with closed-form selection over the input rubric. Anchoring all candidates in the group $\mathcal{G}$ to the same canonical sets removes the duplication, fragmentation, and granularity drift that contaminated the raw audit outputs in (3).

Reward derivation. The two reward channels are simple counts over the verified rubric items:

$$r_i^{(1)} = -|\mathcal{U}_i|, \quad r_i^{(2)} = -|\mathcal{O}_i|. \quad (6)$$

Both rewards are negated so that fewer issues correspond to higher reward, and both are integer-valued, with the unimplemented reward $r_i^{(1)} \in \{-M_{\mathcal{U}}, \dots, 0\}$ bounded by the size of the unimplemented rubric $\mathcal{R}_{\mathcal{G}}^{\mathcal{U}}$, and similarly for the unintended reward $r_i^{(2)} \in \{-M_{\mathcal{O}}, \dots, 0\}$. Because every candidate in the group $\mathcal{G}$ is scored against the same canonical sets $\mathcal{R}_{\mathcal{G}}^{\mathcal{U}}$ and $\mathcal{R}_{\mathcal{G}}^{\mathcal{O}}$, the counts in (6) are directly comparable across the group, which is the property the advantage estimator in Sec. 3.4 relies on.

3.4 Learning Objective

Following GDPO, we instantiate the multi-reward pipeline of Eq. (2) with $n=2$ channels. We z -score $r_i^{(1)}$ and $r_i^{(2)}$ within the group $\mathcal{G}$ to obtain per-channel advantages $A_i^{(1)}$ and $A_i^{(2)}$, combine them linearly into a group-level advantage

$$A_i^{\text{sum}} = w_1 A_i^{(1)} + w_2 A_i^{(2)}, \quad w_1, w_2 > 0, \quad (7)$$

and re-normalize across the training batch $\mathcal{B}$ (all candidates from all groups in the batch),

$$\hat{A}_i = \frac{A_i^{\text{sum}} - \mu_{\mathcal{B}}^{\text{sum}}}{\sigma_{\mathcal{B}}^{\text{sum}} + \varepsilon_0}, \quad (8)$$

with $\mu_{\mathcal{B}}^{\text{sum}}, \sigma_{\mathcal{B}}^{\text{sum}}$ the mean and standard deviation of A^{sum} across $\mathcal{B}$. Per-channel group normalization places the unimplemented and unintended signals on a common, dimensionless scale within each group, so neither dominates the gradient simply by virtue of having larger raw variance, and the relative emphasis between efficacy and locality is controlled by (w_1, w_2) rather than implicitly by reward-scale differences. The batch-level renormalization in Eq. (8) keeps the numerical scale of $\hat{A}_i$ stable across batches and consistent with the single-reward baseline, so the PPO clipping threshold ϵ_{clip} behaves uniformly. Collapsing the two rewards into a single scalar *before* group normalization would forfeit the per-channel scaling property; we show in Sec. 4.4 (variant 2) that doing so degrades training by ~ 1.5 EditScore. Plugging $\hat{A}_i$ into the GDPO PPO-clip objective gives our final training objective $\mathcal{J}_{\text{ours}}(\theta)$ (full form in App. B). Groups with zero within-group variance on both channels supply no learnable ranking and are skipped at the gradient step, but are largely eliminated upstream by the rejection-sampling stage of our data pipeline (App. E.4).

4 Experiments

4.1 Setup

Training data. We use the $\sim 10\text{K}$ -prompt training split of **MinEval**, a dataset we construct by a three-stage VLM pipeline over MagicBrush [Zhang et al., 2023] source images, covering six atomic

edit types (add, remove, move, replace, appearance_change, action_change) and rejection-sampled with the un-tuned base editor to drop unrealizable and trivial prompts; full pipeline, prompts, and checklist are in App. E.

Evaluation benchmarks. We evaluate on four *complementary* benchmarks: (1) the held-out 600-prompt split of **MinEval** (in-distribution; 6 atomic edit types, disjoint from training), (2) **MagicBrush** [Zhang et al., 2023] for human-curated edits from a different pipeline (535 prompts), (3) **AnyBench** [Yu et al., 2024] for breadth across 18 edit types with fine-grained per-type analysis (900 prompts), and (4) **Emu-Edit** [Sheynin et al., 2023] for out-of-distribution robustness at scale (3,589 prompts; 8 task types). Per-type analysis is concentrated on AnyBench, which is the only benchmark with edit-type labels at this granularity.

Metrics. The primary metric is **EditScore-Overall (ES)** [Luo et al., 2025], a Qwen3-VL-8B+LoRA edit-quality judge on $[0, 10]$ ($\uparrow$). We complement it with two off-the-shelf preference scorers from different model families: **HPSv2 (HPS)** [Wu et al., 2023], a CLIP-H based score on $[0, 1]$ ($\uparrow$), and **ImageReward (IR)** [Xu et al., 2023], a BLIP-based real-valued score ($\uparrow$). To measure preservation more directly, we also report **OffTarget-L1** ($\downarrow$), the mean absolute source-to-edit pixel difference outside a SAM 2 target mask on 600 randomly selected examples from the four benchmarks; its definition and scope are in App. C.1. The three EditScore sub-scores (PF, Cons, PQ) are in App. C.4.

Baselines. We compare against six external instruction-edit models InstructPix2Pix [Brooks et al., 2023], MagicBrush [Zhang et al., 2023], OmniGen2 [Wu et al., 2025b], UniWorld-v1 [Lin et al., 2025a], ICEdit [Zhang et al., 2025], BAGEL [Deng et al., 2025] spanning $\sim 1\text{B}$ – 20B parameters and UNet/DiT/MoT families (sizes annotated in Table 1). Our backbone reference is the un-tuned FLUX.1 Kontext-dev [Black Forest Labs et al., 2025] (12B), which isolates the gain attributable to RL fine-tuning. All methods use identical inference settings (Sec. 4.2).

4.2 Implementation Details

Training. We fine-tune FLUX.1 Kontext-dev [Black Forest Labs et al., 2025] (12B) with Mix-GRPO [Li et al., 2025] as the RL backbone, training for 230 steps (≈ 300 GPU-hr on $8 \times \text{A100-80GB}$). The reward is produced by Qwen3-VL-30B-A3B-Instruct [Qwen Team, 2025] as the audit/verification VLM Φ_V and Qwen3-30B-A3B-Instruct-2507 [Qwen Team, 2025] as the rubric-aggregation LLM Φ_L , both served via vLLM [Kwon et al., 2023]. Full hyperparameters are in App. D.

Reward-generation cost. For a group of 16 candidates, audit, aggregation, and verification took 16.06 seconds and processed 61,950 tokens under the hardware and serving configuration in App. C.1. This measures reward generation rather than end-to-end training throughput.

Evaluation. All methods are evaluated under identical inference settings (resolution, denoising steps, CFG, seed; details in App. D); metrics follow Sec. 4.1.

4.3 Main Results

Cross-benchmark comparison. ARRO achieves the highest unweighted average across ES, HPSv2, and IR (Table 1), with gains over the un-tuned base on all four benchmarks. BAGEL leads on MinEval and MagicBrush ES, while ARRO leads on AnyBench and Emu-Edit ES; thus the four-benchmark average favors ARRO (5.88 vs. 5.80). The independent scorers provide complementary quality evidence, while the masked pixel analysis below directly tests off-target preservation.

Off-target modification. On the 600-example masked evaluation, ARRO reduces OffTarget-L1 from 0.20366 (Base) to 0.18659, an 8.4% relative reduction. The EditScore(PF, Cons)-GDPO reward control and a separate EditScore-Top1-SFT control obtain 0.18975 and 0.19952, respectively (App. C.1). This result directly supports reduced change outside the designated target mask; it does not imply that every pixel inside the mask was correctly edited or that segmentation is error-free.

Zero-shot generalization across edit semantics (AnyBench). ARRO is trained on only six atomic edit types, but AnyBench’s 18-category taxonomy lets us probe its generalization to unseen edit semantics (Table 2). The improvements are systemic rather than category-bound: ARRO beats the un-tuned base on 16/18 categories under both EditScore and HPSv2, with the largest gains on tasks where locality is notoriously difficult, e.g., movement (+1.58 ES), *implicit_chg.* (+1.30), and

Table 1: Final-checkpoint results on four image-editing benchmarks. We report EDITSCORE (ES) Overall per benchmark, plus the unweighted mean across benchmarks for all three metrics (**Avg**, $\text{mean}_{\pm\text{std}}$ across the four benchmarks). The four benchmarks differ in size by $\sim 6\times$, so we use unweighted averaging to give equal voice to each benchmark; per-benchmark HPSv2 and ImageReward are in App. C.5. Higher is better in all columns; $\text{ES} \in [0, 10]$, $\text{HPS} \in [0, 1]$, IR is real-valued. Best per column in **bold**.

Method	Avg			ES per benchmark			
	ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$	MinEval	MagicBr.	AnyBench	Emu-Edit
FLUX.1 Kontext-dev (Base, 12B, 2025)	5.21 ± 0.34	0.215 ± 0.007	-0.67 ± 0.14	4.72	5.25	5.46	5.40
InstructPix2Pix ($\sim 1\text{B}$, 2022)	2.43 ± 0.34	0.211 ± 0.009	-0.89 ± 0.17	1.93	2.50	2.63	2.65
MagicBrush model ($\sim 1\text{B}$, 2023)	3.53 ± 0.63	0.207 ± 0.006	-0.90 ± 0.07	3.25	4.45	3.04	3.38
OmniGen2 ($\sim 7\text{B}$, 2025)	5.08 ± 0.30	0.214 ± 0.006	-0.72 ± 0.10	5.15	5.03	4.71	5.44
UniWorld-v1 ($\sim 20\text{B}$, 2025)	3.43 ± 0.26	0.202 ± 0.004	-0.92 ± 0.05	3.38	3.40	3.78	3.15
ICEdit ($\sim 12\text{B}$, 2025)	5.15 ± 0.57	0.207 ± 0.006	-0.81 ± 0.08	4.90	5.73	4.48	5.50
BAGEL ($\sim 14\text{B}$, 2025)	5.80 ± 0.43	0.209 ± 0.003	-0.73 ± 0.10	5.98	6.22	5.22	5.79
ARRO (12B, 2026, ours)	5.88 ± 0.15	0.222 ± 0.008	-0.51 ± 0.16	5.73	5.80	6.07	5.91

Table 2: Per-edit-type EDITSCORE Overall on AnyBench (50 prompts per edit type). Edit-type names follow AnyBench’s taxonomy and may differ in surface form from the six training edit types in App. E.1 (e.g., `appearance_alter` vs. `appearance_change`, `action_chg.` vs. `action_change`). Edit types are columns and methods are rows. Higher is better. The best score in each column is shown in **bold**, and the second-best score is marked with *. Full results are in App. C.6.

Method	Edit type																
	act.	add	app.	bg.	color	count	impl.	mat.	move	outp.	rel.	rem.	repl.	resize	rot.	style	text
Base	5.76*	7.88	5.74	6.29	5.62	4.51*	5.09*	4.18	2.51	4.71*	3.80	5.85*	5.87	3.98*	7.10	5.37*	8.34
ARRO	6.69	8.33*	6.45*	6.77	6.25	5.03	6.39	4.57*	4.09	4.89	4.60	5.62	6.41*	5.00	7.47*	5.92	8.25*
IP2P	1.48	4.62	3.02	2.89	4.99	0.42	2.72	2.61	1.95	2.48	2.35	1.46	2.50	2.19	3.02	3.42	0.08
MagicBr.	2.39	6.98	4.78	3.06	3.78	1.79	3.77	2.15	3.29	1.20	3.31	3.45	3.61	1.55	2.43	2.22	0.86
OmniGen2	2.56	7.18	5.93	6.72*	6.46	2.32	4.42	3.98	2.16	4.62	4.02*	6.30	6.74	1.61	2.70	4.55	6.78
UniWorld-v1	4.38	6.09	3.88	4.20	4.76	3.48	4.93	3.49	2.95	3.59	3.60	3.84	4.46	1.90	4.51	4.14	0.96
ICEdit	2.40	8.15	5.98	5.74	6.63*	2.15	3.91	4.60	2.66	2.62	3.13	4.80	5.11	2.47	2.93	5.03	5.78
BAGEL	4.57	8.86	6.62	5.73	6.79	3.67	4.74	3.78	3.28*	2.82	3.35	5.61	5.78	2.84	8.12	4.25	7.34

`resize` (+1.02). This breadth suggests the dual-channel reward pushes the policy to learn localized image manipulation rather than memorizing specific instruction templates.

4.4 Ablations

ARRO combines two critical design choices on top of GDPO [Liu et al., 2026]: a *dual-channel* reward ($r^{(1)}, r^{(2)}$) that decouples efficacy from unintended change, and *per-channel group normalization* that turns the two raw reward channels into commensurable advantages ($A^{(1)}, A^{(2)}$) before they are combined. Table 3 enumerates the variants we compare; each isolates a single design factor. All variants share identical backbones, training data, reward models, and hyperparameters, evaluated at the 230-step final checkpoint.

Table 3: Ablation variants and results. The **Reward** column gives the reward signal used for RL fine-tuning; variants (2)–(4) toggle one design factor relative to the full method (5). **Avg** is the unweighted mean across the four benchmarks. Final-checkpoint EditScore Overall ($\uparrow$), HPSv2 ($\uparrow$), ImageReward ($\uparrow$); best per column in **bold**. Variant (4) reward-hacks to a degenerate noise output and is collapsed to a single line (see text).

Variant	Reward	Avg			MinEval			MagicBrush			AnyBench			Emu-Edit		
		ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$	ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$	ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$	ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$	ES $\uparrow$	HPS $\uparrow$	IR $\uparrow$
(1) Base (no RL)	$\emptyset$	5.21 ± 0.34	0.215 ± 0.007	-0.67 ± 0.14	4.72	0.212	-0.80	5.25	0.224	-0.51	5.46	0.215	-0.58	5.40	0.208	-0.78
(2) Merged scalar	$w_1 r^{(1)} + w_2 r^{(2)}$	4.35 ± 0.37	0.211 ± 0.006	-0.75 ± 0.10	3.89	0.209	-0.82	4.23	0.219	-0.62	4.75	0.212	-0.72	4.52	0.205	-0.85
(3) Efficacy only	$r^{(1)}$	4.09 ± 0.45	0.203 ± 0.007	-0.87 ± 0.17	3.46	0.197	-1.05	4.09	0.212	-0.65	4.49	0.204	-0.83	4.33	0.197	-0.94
(4) Locality only [reward-hacked]	$r^{(2)}$	ES ≈ 0.00 HPS ≈ 0.011 IR ≈ -2.25 across all four benchmarks (degenerate noise output)														
(5) ARRO (full, ours)	$(r^{(1)}, r^{(2)})$ + per-channel adv.	5.88 ± 0.15	0.222 ± 0.008	-0.51 ± 0.16	5.73	0.220	-0.63	5.80	0.233	-0.29	6.07	0.221	-0.49	5.91	0.214	-0.62

Table 4: Blinded pairwise judgments on 60 MinEval examples by ten evaluators (600 judgments per comparison). Win, tie, and loss are from the ARRO variant’s perspective; judgments share 60 underlying image pairs.

Comparison	Win	Tie	Loss
ARRO vs. EditScore(PF, Cons)-GDPO	366/600 (61.0%)	94/600 (15.7%)	140/600 (23.3%)
OmniGen2 + ARRO vs. OmniGen2	321/600 (53.5%)	107/600 (17.8%)	172/600 (28.7%)

Table 5: OmniGen2 transfer on three unseen benchmarks. EditScore (ES) is higher-is-better; the average is unweighted. Full HPSv2 and ImageReward results are in App. C.3.

Benchmark	OmniGen2	OmniGen2 + ARRO	ES gain
MagicBrush	5.03	5.58	+0.55
AnyBench	4.71	5.22	+0.51
Emu-Edit	5.44	5.99	+0.55
Average	5.06	5.59	+0.53

Summary. Each ablation exposes a distinct failure mode. *Merged scalar (2)*: collapsing the dual channels before group normalization drops EditScore by ~ 1.5 , as the higher-variance signal washes out the other, confirming per-channel normalization is essential. *Efficacy only (3)*: without locality pressure, the policy over-edits and corrupts non-target context (-1.12 ES vs. Base). *Locality only (4)*: by step ~ 170 the policy reward-hacks to high-frequency noise, $|\mathcal{O}_i| \rightarrow 0$, and $r^{(2)}$ saturates even though every instruction is unimplemented. The dual-channel design (5) closes both shortcuts: $r^{(1)}$ anchors semantic validity so neither over-editing nor empty outputs is rewarded.

Reward construction and optimization controls. With GDPO and the training configuration fixed, EditScore(PF, Cons) rewards yield 3.777 average ES across the four benchmarks, versus 5.880 for ARRO rewards. Removing aggregation while retaining verification reduces the four-benchmark average ES from 5.880 to 4.961, supporting the value of a shared, consolidated rubric. In a separate MinEval-only comparison, raw audit counts yield 3.2779 ES, ARRO Top-10 SFT yields 4.6289, and ARRO-GDPO yields 5.6690; these values are not four-benchmark averages. In a controlled CLEVR diagnostic with all requested recolorings implemented, ARRO assigns zero unintended changes for 5–50 requested edits, whereas EditScore Consistency varies despite zero ground-truth unintended changes (App. C.1). Detailed protocols and metrics are in App. C.1.

Human validation and backbone transfer. Two blinded annotators checked 60 candidate groups (960 candidates) and 565 final issues: they jointly confirmed correct merging for 560/565 issues, visual support for 553/565, and no evident omitted issue in 58/60 groups. In a separate blinded pairwise study, ARRO wins more comparisons than it loses against EditScore(PF, Cons)-GDPO (Table 4). Applying the same reward pipeline to OmniGen2 with its FlowGRPO optimizer improves ES on all three unseen benchmarks (Table 5) and wins more human comparisons than the original OmniGen2 (Table 4). Study protocols and full transfer metrics are in App. C.2 and App. C.3.

5 Conclusion

We recast *minimal change* for instruction-based image editing as an outcome-level RL objective with separate rewards for unimplemented and unintended changes. On FLUX.1 Kontext-dev, ARRO improves average EditScore from 5.21 to 5.88 across four benchmarks and reduces masked off-target pixel change by 8.4%. Reward and SFT controls, human evaluations, and OmniGen2 transfer support the value of structured reward construction and direct optimization.

Limitations and broader impacts. Reward quality is bounded by the VLM’s perception, especially for subtle differences, ambiguous or compositional instructions, and candidate-specific issue assignment. An imperfect judge can introduce systematic errors or be exploited during RL; the expert study validates the evaluated configuration rather than all possible auditors. OffTarget-L1 depends on segmentation and captures pixel change rather than full semantic correctness. We demonstrate a second rectified-flow editor, OmniGen2, but do not establish transfer to diffusion backbones or quantify

multi-seed variability. More controllable editing could also facilitate deceptive image manipulation; deployment requires provenance and misuse safeguards beyond the reward mechanism studied here.

6 Acknowledgments

Tongliang Liu is partially supported by the following Australian Research Council projects: DP260102466, FT220100318, DP220102121, LP220100527, LP220200949. Yu Yao is partially supported by the following Australian Research Council projects: DE260101993, DP260102466.

References

- Elior Benarous, Yilun Du, and Heng Yang. SPIE: Semantic and structural post-training of image editing diffusion models with AI feedback. *arXiv preprint arXiv:2504.12833*, 2025.
- 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. *arXiv preprint arXiv:2506.15742*, 2025.
- 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 (CVPR)*, pages 18392–18402, 2023.
- Aleksandar Cvejjic, Rameen Abdal, Abdelrahman Eldesokey, Bernard Ghanem, and Peter Wonka. NearID: Identity representation learning via near-identity distractors, 2026.
- Chaorui Deng, Deyao Zhu, Kunchang Li, Chenhui Gou, Feng Li, Zeyu Wang, Shu Zhong, Weihao Yu, Xiaonan Nie, Ziang Song, Guang Shi, and Haoqi Fan. Emerging properties in unified multimodal pretraining. *arXiv preprint arXiv:2505.14683*, 2025.
- Connor Dunlop, Matthew Zheng, Kavana Venkatesh, and Pinar Yanardag. Personalized image editing in text-to-image diffusion models via collaborative direct preference optimization. *arXiv preprint arXiv:2511.05616*, 2025.
- Yuan Gong, Xionghui Wang, Jie Wu, Shiyin Wang, Yitong Wang, and Xinglong Wu. OneReward: Unified mask-guided image generation via multi-task human preference learning. *arXiv preprint arXiv:2508.21066*, 2025.
- Ziming Hong, Tianyu Huang, Runnan Chen, Shanshan Ye, Mingming Gong, Bo Han, and Tongliang Liu. AdLift: Lifting adversarial perturbations to safeguard 3D gaussian splatting assets against instruction-driven editing. *arXiv preprint arXiv:2512.07247*, 2025. URL <https://arxiv.org/abs/2512.07247>.
- Yuxin Huang, Ziming Hong, Mingming Gong, Wanyu Wang, Jing Zhang, and Tongliang Liu. Delving into the temporal challenges of unified video protection against image-to-video and fine-tuning-based customization. *arXiv preprint arXiv:2607.13336*, 2026. URL <https://arxiv.org/abs/2607.13336>.
- Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, and Ying Shan. SmartEdit: Exploring complex instruction-based image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. arXiv:2312.06739.
- Bohan Jia, Wenxuan Huang, Yuntian Tang, Junbo Qiao, Jincheng Liao, Shaosheng Cao, Fei Zhao, Zhaopeng Feng, Zhouhong Gu, Zhenfei Yin, Lei Bai, Wanli Ouyang, Lin Chen, Fei Zhao, Yao Hu, Zihan Wang, Yuan Xie, and Shaohui Lin. CompBench: Benchmarking complex instruction-guided image editing. *arXiv preprint arXiv:2505.12200*, 2025.
- Zhangqi Jiang, Zheng Sun, Xianfang Zeng, Yufeng Yang, Xuanyang Zhang, Yongliang Wu, Wei Cheng, Gang Yu, Xu Yang, and Bihan Wen. GEditBench v2: A human-aligned benchmark for general image editing. *arXiv preprint arXiv:2603.28547*, 2026.

- Yutao Jin, Yuang Tao, and Junyong Zhai. FlexCausal: Flexible causal disentanglement via structural flow priors and manifold-aware interventions. *arXiv preprint arXiv:2601.21567*, 2026.
- Bowen Jing, Ruiyang Hao, Weitao Zhou, and Haibao Yu. CounterScene: Counterfactual causal reasoning in generative world models for safety-critical closed-loop evaluation. *arXiv preprint arXiv:2603.21104*, 2026.
- Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.
- Ronen Kamenetsky, Sara Dorfman, Daniel Garibi, Roni Paiss, Or Patashnik, and Daniel Cohen-Or. SAEEdit: Token-level control for continuous image editing via sparse autoencoder. *arXiv preprint arXiv:2510.05081*, 2025.
- Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. arXiv:1912.04958.
- Aneesh Komanduri, Yongkai Wu, Wen Huang, Feng Chen, and Xintao Wu. SCM-VAE: Learning identifiable causal representations via structural knowledge. In *2022 IEEE International Conference on Big Data (Big Data)*, pages 1014–1023, 2022. doi: 10.1109/BigData55660.2022.10021114.
- Aneesh Komanduri, Chen Zhao, Feng Chen, and Xintao Wu. Causal diffusion autoencoders: Toward counterfactual generation via diffusion probabilistic models. *arXiv preprint arXiv:2404.17735*, 2024.
- Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023.
- Sébastien Lachapelle, Pau Rodríguez López, Yash Sharma, Katie Everett, Rémi Le Priol, Alexandre Lacoste, and Simon Lacoste-Julien. Nonparametric partial disentanglement via mechanism sparsity: Sparse actions, interventions and sparse temporal dependencies. *arXiv preprint arXiv:2401.04890*, 2024.
- Hengjia Li, Liming Jiang, Qing Yan, Yizhi Song, Hao Kang, Zichuan Liu, Xin Lu, Boxi Wu, and Deng Cai. ThinkRL-Edit: Thinking in reinforcement learning for reasoning-centric image editing. *arXiv preprint arXiv:2601.03467*, 2026.
- Junzhe Li, Yutao Cui, Tao Huang, Yinping Ma, Chun Fan, Miles Yang, and Zhao Zhong. MixGRPO: Unlocking flow-based GRPO efficiency with mixed ODE-SDE, 2025. URL <https://arxiv.org/abs/2507.21802>.
- Shufan Li, Harkanwar Singh, and Aditya Grover. InstructAny2Pix: Flexible visual editing via multimodal instruction following. *arXiv preprint arXiv:2312.06738*, 2023.
- Bin Lin, Zongjian Li, Xinhua Cheng, Yuwei Niu, Yang Ye, Xianyi He, Shenghai Yuan, Wangbo Yu, Shaodong Wang, Yuniang Ge, Yatian Pang, and Li Yuan. Uniworld-v1: High-resolution semantic encoders for unified visual understanding and generation, 2025a. URL <https://arxiv.org/abs/2506.03147>.
- Yexiong Lin, Yu Yao, and Tongliang Liu. Beyond optimal transport: Model-aligned coupling for flow matching. *arXiv preprint arXiv:2505.23346*, 2025b. URL <https://arxiv.org/abs/2505.23346>.
- Yexiong Lin, Jia Shi, Shanshan Ye, Wanyu Wang, Yu Yao, and Tongliang Liu. SubFlow: Sub-mode conditioned flow matching for diverse one-step generation. *arXiv preprint arXiv:2604.12273*, 2026. URL <https://arxiv.org/abs/2604.12273>.

- Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-GRPO: Training flow matching models via online RL. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2025. arXiv:2505.05470.
- Shih-Yang Liu, Xin Dong, Ximing Lu, Shizhe Diao, Peter Belcak, Mingjie Liu, Min-Hung Chen, Hongxu Yin, Yu-Chiang Frank Wang, Kwang-Ting Cheng, Yejin Choi, Jan Kautz, and Pavlo Molchanov. GDPO: Group reward-decoupled normalization policy optimization for multi-reward RL optimization. *arXiv preprint arXiv:2601.05242*, 2026.
- Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. *arXiv preprint arXiv:2209.03003*, 2022.
- Yancheng Long, Yankai Yang, Hongyang Wei, Wei Chen, Tianke Zhang, Haonan fan, Changyi Liu, Kaiyu Jiang, Jiankang Chen, Kaiyu Tang, Bin Wen, Fan Yang, Tingting Gao, Han Li, and Shuo Yang. SpatialReward: Bridging the perception gap in online RL for image editing via explicit spatial reasoning. *arXiv preprint arXiv:2602.07458*, 2026.
- Romain Lopez, Nataša Tagasovska, Stephen Ra, Kyunghyun Cho, Jonathan K. Pritchard, and Aviv Regev. Learning causal representations of single cells via sparse mechanism shift modeling. In *Conference on Causal Learning and Reasoning (CLearR)*, 2023. arXiv:2211.03553.
- Xin Luo, Jiahao Wang, Chenyuan Wu, Shitao Xiao, Xiyan Jiang, Defu Lian, Jiajun Zhang, Dong Liu, and Zheng liu. EditScore: Unlocking online RL for image editing via high-fidelity reward modeling. *arXiv preprint arXiv:2509.23909*, 2025.
- Weizhi Nie, Zichun Zhang, Weijie Wang, Bruno Lepri, Anan Liu, and Nicu Sebe. Causal disentanglement for robust long-tail medical image generation. *arXiv preprint arXiv:2504.14450*, 2025.
- Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, Cambridge, UK, 2nd edition, 2009.
- Ronan Perry, Julius von Kügelgen, and Bernhard Schölkopf. Causal discovery in heterogeneous environments under the sparse mechanism shift hypothesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pages 10904–10917, 2022.
- Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press, Cambridge, MA, 2017.
- Yusu Qian, Jiasen Lu, Tsu-Jui Fu, Xinze Wang, Chen Chen, Yinfei Yang, Wenze Hu, and Zhe Gan. GIE-Bench: Towards grounded evaluation for text-guided image editing. *arXiv preprint arXiv:2505.11493*, 2025.
- Qwen Team. Qwen3 technical report, 2025. Model card: Qwen/Qwen3-30B-A3B-Instruct-2507.
- Rajat Rasal, Avinash Kori, Fabio De Sousa Ribeiro, Tian Xia, and Ben Glocker. Diffusion counterfactual generation with semantic abduction. *arXiv preprint arXiv:2506.07883*, 2025.
- Nikhila Ravi et al. SAM 2: Segment anything in images and videos, 2024.
- Juan Rocamonde, Victoriano Montesinos, Elvis Nava, Ethan Perez, and David Lindner. Vision-language models are zero-shot reward models for reinforcement learning. *arXiv preprint arXiv:2310.12921*, 2023.
- Bernhard Schölkopf, Francesco Locatello, Stefan Bauer, Nan Rosemary Ke, Nal Kalchbrenner, Anirudh Goyal, and Yoshua Bengio. Toward causal representation learning. *Proceedings of the IEEE*, 109(5):612–634, 2021. doi: 10.1109/JPROC.2021.3058954.
- Mutiara Shabrina, Nova Kurnia Putri, Jefri Satria Ferdiansyah, Sabita Khansa Dewi, and Novanto Yudistira. Training-free disentangled text-guided image editing via sparse latent constraints. *arXiv preprint arXiv:2512.21637*, 2025.
- Xinwei Shen, Furui Liu, Hanze Dong, Qing Lian, Zhitang Chen, and Tong Zhang. Weakly supervised disentangled generative causal representation learning. *Journal of Machine Learning Research*, 23(241):1–55, 2022.

- Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of GANs for semantic face editing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. arXiv:1907.10786.
- Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu Edit: Precise image editing via recognition and generation tasks. *arXiv preprint arXiv:2311.10089*, 2023.
- Bram Wallace, Meihua Dang, Rafael Rafailov, Linqi Zhou, Aaron Lou, Senthil Purushwalkam, Stefano Ermon, Caiming Xiong, Shafiq Joty, and Nikhil Naik. Diffusion model alignment using direct preference optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024. arXiv:2311.12908.
- Qian Wang, Aleksandar Cvejjic, Abdelrahman Eldesokey, and Peter Wonka. EditCLIP: Representation learning for image editing. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025. doi: 10.1109/ICCV51701.2025.01481.
- 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. *arXiv preprint arXiv:2508.02324*, 2025a.
- Chenyuan Wu, Pengfei Zheng, Ruirao Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, Ze Liu, Ziyi Xia, Chaofan Li, Haoge Deng, Jiahao Wang, Kun Luo, Bo Zhang, Defu Lian, Xinlong Wang, Zhongyuan Wang, Tiejun Huang, and Zheng Liu. OmniGen2: Towards instruction-aligned multimodal generation. *arXiv preprint arXiv:2506.18871*, 2025b.
- Jie Wu, Yu Gao, Zilyu Ye, Ming Li, Liang Li, Hanzhong Guo, Jie Liu, Zeyue Xue, Xiaoxia Hou, Wei Liu, Yan Zeng, and Weilin Huang. RewardDance: Reward scaling in visual generation. *arXiv preprint arXiv:2509.08826*, 2025c.
- Mingyuan Wu, Jingcheng Yang, Jize Jiang, Meitang Li, Kaizhuo Yan, Hanchao Yu, Minjia Zhang, Chengxiang Zhai, and Klara Nahrstedt. VTool-R1: VLMs learn to think with images via reinforcement learning on multimodal tool use. *arXiv preprint arXiv:2505.19255*, 2025d.
- Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis. *arXiv preprint arXiv:2306.09341*, 2023.
- Yongli Xiang, Ziming Hong, Zhaoqing Wang, Xiangyu Zhao, Bo Han, and Tongliang Liu. When safety collides: Resolving multi-category harmful conflicts in text-to-image diffusion via adaptive safety guidance. *arXiv preprint arXiv:2602.20880*, 2026. URL <https://arxiv.org/abs/2602.20880>.
- Danru Xu, Dingling Yao, Sébastien Lachapelle, Perouz Taslakian, Julius von Kügelgen, Francesco Locatello, and Sara Magliacane. A sparsity principle for partially observable causal representation learning. *arXiv preprint arXiv:2403.08335*, 2024.
- Jiazhen Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. ImageReward: Learning and evaluating human preferences for text-to-image generation. *arXiv preprint arXiv:2304.05977*, 2023.
- Zeyue Xue, Jie Wu, Yu Gao, Fangyuan Kong, Lingting Zhu, Mengzhao Chen, Zhiheng Liu, Wei Liu, Qiushan Guo, Weilin Huang, et al. DanceGRPO: Unleashing GRPO on visual generation. *arXiv preprint arXiv:2505.07818*, 2025.
- Mengyue Yang, Furui Liu, Zhitang Chen, Xinwei Shen, Jianye Hao, and Jun Wang. CausalVAE: Disentangled representation learning via neural structural causal models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9593–9602, 2021.

- Yu Yao, Huanjian Zhou, Andi Han, Wei Huang, and Masashi Sugiyama. Accelerating discrete diffusion models with parallel-in-time sampling. *arXiv preprint arXiv:2607.00773*, 2026. URL <https://arxiv.org/abs/2607.00773>.
- Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang Tang, Hanwang Zhang, and Yueting Zhuang. AnyEdit: Mastering unified high-quality image editing for any idea. *arXiv preprint arXiv:2411.15738*, 2024.
- Kwan Yun, Changmin Lee, Ayeong Jeong, Youngseo Kim, Seungmi Lee, and Junyong Noh. StyleID: A perception-aware dataset and metric for stylization-agnostic facial identity recognition, 2026.
- Kai Zhang, Lingbo Mo, Wenhui Chen, Huan Sun, and Yu Su. MagicBrush: A manually annotated dataset for instruction-guided image editing. *arXiv preprint arXiv:2306.10012*, 2023.
- Zechuan Zhang, Ji Xie, Yu Lu, Zongxin Yang, and Yi Yang. In-context edit: Enabling instructional image editing with in-context generation in large scale diffusion transformer. *arXiv preprint arXiv:2504.20690*, 2025.
- Hanyang Zhao, Haoxian Chen, Yucheng Guo, Genta Indra Winata, Tingting Ou, Ziyu Huang, David D. Yao, and Wenpin Tang. RPO: Fine-tuning visual generative models via rich vision-language preferences. *arXiv preprint arXiv:2503.11720*, 2025.
- Bowen Zheng, Yongli Xiang, Ziming Hong, Zerong Lin, Chaojian Yu, Tongliang Liu, and Xinge You. VII: Visual instruction injection for jailbreaking image-to-video generation models. *arXiv preprint arXiv:2602.20999*, 2026. URL <https://arxiv.org/abs/2602.20999>.

A Related Work (Detailed)

Instruction-based image editing: pretraining. The dominant paradigm for instruction-based image editing trains a conditional generative model on large-scale (source, instruction, edit) triplets. InstructPix2Pix [Brooks et al., 2023] pioneered this direction by synthesizing paired training data from a language model and a text-to-image model, enabling a forward-pass diffusion editor to follow free-form instructions without per-example optimization. Subsequent work scaled this recipe along two axes: stronger language understanding through multimodal LLM backbones [Huang et al., 2024, Li et al., 2023], and stronger visual backbones through diffusion-transformer architectures and rectified flow, most notably FLUX.1 Kontext [Black Forest Labs et al., 2025] and Qwen-Image-Edit [Wu et al., 2025a], which explicitly target locality and minimal background drift as a design goal. Across this line of work, locality is treated as an emergent property of data quality, conditioning design, and model capacity; it is not optimized for directly. As reported by recent benchmarks [Jia et al., 2025, Qian et al., 2025, Jiang et al., 2026], even the strongest pretrained editors still leak edits into the background, shift identity, or introduce unintended attribute changes, motivating a post-training stage that targets these failures explicitly.

Instruction-based image editing: post-training. A second line of work post-trains pretrained editors using preference or reward feedback. Diffusion-DPO [Wallace et al., 2024] adapts direct preference optimization to diffusion models, with follow-up work extending it to richer or per-region preference signals [Zhao et al., 2025, Dunlop et al., 2025, Benarous et al., 2025]. On the online-RL side, Flow-GRPO [Liu et al., 2025] and DanceGRPO [Xue et al., 2025] bring group-relative policy optimization to flow-matching generators, and GDPO [Liu et al., 2026] extends this to multiple rewards. For the editing setting specifically, EditScore [Luo et al., 2025], OneReward [Gong et al., 2025], RewardDance [Wu et al., 2025c], SpatialReward [Long et al., 2026], and ThinkRL-Edit [Li et al., 2026] train dedicated reward models to drive online RL fine-tuning. Our work differs from this line on two fronts. *First, human preference is not minimal change.* Prior rewards fit human preference data, which mixes many factors at once: visual quality, instruction adherence, aesthetic appeal, and overall plausibility. Whether the edit altered things outside the intended scope is only one of these factors, often the least salient one to a human rater glancing at the result. A reward trained to predict human preference therefore has no reason to penalize unintended changes specifically, especially when the edit otherwise looks plausible. We target minimal change directly instead. *Second, prior rewards are not scalable.* Each of EditScore, OneReward, RewardDance, SpatialReward, and ThinkRL-Edit is a learned reward model, requiring tens to hundreds of thousands of annotated

preference pairs and a training pipeline of its own. Our reward requires no preference data and no reward-model training: an off-the-shelf VLM is prompted to audit each edit, and the two reward components for minimal change are read off directly from its output.

Minimal change in causal representation learning. The principle that an intervention should perturb only the targeted mechanism originates in causal inference, where it underpins Pearl’s *do*-calculus and the structural-causal-model account of counterfactuals [Pearl, 2009, Peters et al., 2017]. In representation learning, this principle is formalized as the *sparse mechanism shift* hypothesis [Schölkopf et al., 2021, Perry et al., 2022, Lopez et al., 2023]: distribution shifts across environments correspond to sparse changes in a small number of causal mechanisms, and exploiting this sparsity yields identifiability of the latent causal factors [Lachapelle et al., 2024, Xu et al., 2024]. Within deep generative models, CausalVAE [Yang et al., 2021], DEAR [Shen et al., 2022], SCM-VAE [Komanduri et al., 2022], and more recent flow-prior variants [Jin et al., 2026, Komanduri et al., 2024] operationalize minimal change by structuring the latent space around a learned causal graph and pairing it with weak label or counterfactual supervision, so that interventions can be directed at individual latent factors. A parallel, training-free line of work in StyleGAN-era image editing makes the same intuition explicit by adding L_1 or sparse-autoencoder constraints to the latent edit at inversion time [Shabrina et al., 2025, Kamenetsky et al., 2025]. Our work follows the same conceptual goal but disagrees on where the constraint should live: we argue that minimal change is fundamentally a property of the *output* of an intervention, and that enforcing it on a latent that is then passed through a deeply nonlinear decoder is a brittle proxy.

Minimal change for image generation and editing. The same principle has been brought to image-level tasks via counterfactual image generation under structural causal models [Komanduri et al., 2024, Rasal et al., 2025, Nie et al., 2025, Jing et al., 2026] and StyleGAN-era disentangled face editing with axis-aligned latents or training-free L_1 /layer-masking constraints [Shabrina et al., 2025, Kamenetsky et al., 2025]. These methods deliver strong locality on narrow domains with attribute labels or counterfactual pairs, but inherit the same three obstacles we identify in Sec. 1 when scaled to free-form instructions; we sidestep all three by measuring minimal change on the generated image and optimizing it end-to-end via an agentic reward.

B Optimization Details

This section spells out the GDPO optimizer and our dual-channel advantage in full, complementing the abbreviated description in Sec. 3.1 and Sec. 3.4.

Rectified-flow training objective. We restate Eq. (1) for reference. FLUX.1 Kontext is pretrained as a rectified-flow transformer [Liu et al., 2022, Black Forest Labs et al., 2025]. With $z_0 \sim p_{\text{data}}$, $\epsilon \sim \mathcal{N}(0, I)$, $t \sim \mathcal{U}[0, 1]$, and source-image latent z_{src} concatenated with the noisy latent $z_t = (1-t)z_0 + t\epsilon$ along the sequence axis, the velocity field v_θ is trained with

$$\mathcal{L}_{\text{RF}} = \mathbb{E}_{t, z_0, \epsilon} \|v_\theta([z_t; z_{\text{src}}], t, c) - (\epsilon - z_0)\|_2^2. \quad (9)$$

The pretraining loss (9) is not used during our RL fine-tuning; we include it only to fix notation.

GDPO advantage. For an input pair $(x_{\text{src}}, c) \sim \mathcal{D}$, GDPO [Liu et al., 2026] samples a group $\mathcal{G} = \{x_1, \dots, x_N\}$ of N candidates from the current policy $\pi_{\theta_{\text{old}}}(\cdot | x_{\text{src}}, c)$ and scores each candidate by n reward channels $\{r_i^{(k)}\}_{k=1}^n$. The advantage used in policy optimization is constructed in three steps. *Step 1 (per-reward group z -score)*. Each channel is normalized within the group,

$$A_i^{(k)} = \frac{r_i^{(k)} - \mu_{\mathcal{G}}^{(k)}}{\sigma_{\mathcal{G}}^{(k)} + \varepsilon_0}, \quad \mu_{\mathcal{G}}^{(k)} = \frac{1}{N} \sum_{j=1}^N r_j^{(k)}, \quad \sigma_{\mathcal{G}}^{(k)} = \sqrt{\frac{1}{N} \sum_{j=1}^N (r_j^{(k)} - \mu_{\mathcal{G}}^{(k)})^2}, \quad (10)$$

where ε_0 is a small numerical-stability offset. *Step 2 (weighted sum across channels)*. The per-channel advantages are aggregated into a group-level advantage

$$A_i^{\text{sum}} = \sum_{k=1}^n w_k A_i^{(k)}, \quad w_k > 0. \quad (11)$$

Step 3 (batch-level z -score). Finally, A_i^{sum} is renormalized across the training batch $\mathcal{B}$ (all candidates from all groups in $\mathcal{B}$) to keep its scale stable as the number of channels n grows:

$$\hat{A}_i = \frac{A_i^{\text{sum}} - \mu_{\mathcal{B}}^{\text{sum}}}{\sigma_{\mathcal{B}}^{\text{sum}} + \varepsilon_0}, \quad \mu_{\mathcal{B}}^{\text{sum}} = \text{mean}_{j \in \mathcal{B}} A_j^{\text{sum}}, \quad \sigma_{\mathcal{B}}^{\text{sum}} = \text{std}_{j \in \mathcal{B}} A_j^{\text{sum}}. \quad (12)$$

Step 3 is an affine, batch-wide transform: it preserves within-group rankings (and therefore the relative ordering used by PPO) but stabilizes the magnitude that the clip threshold ϵ_{clip} sees.

MixGRPO sampling. Sampling from π_{θ} uses an SDE discretization of the rectified flow [Li et al., 2025] over T denoising steps indexed $t = T, \dots, 1$ (a discrete index, distinct from the continuous flow time of Eq. (9)), inducing transition kernels $\pi_{\theta}(\mathbf{x}_{i,t-1} | \mathbf{x}_{i,t})$ along each candidate’s trajectory. With the per-step importance ratio

$$\rho_{i,t}(\theta) = \frac{\pi_{\theta}(\mathbf{x}_{i,t-1} | \mathbf{x}_{i,t})}{\pi_{\theta_{\text{old}}}(\mathbf{x}_{i,t-1} | \mathbf{x}_{i,t})}, \quad (13)$$

the policy is updated by the PPO-clip objective

$$\mathcal{J}_{\text{GDPO}}(\theta) = \mathbb{E}_{i,t} \left[\min(\rho_{i,t}(\theta) \hat{A}_i, \text{clip}(\rho_{i,t}(\theta), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}) \hat{A}_i) \right], \quad (14)$$

where $\hat{A}_i$ is the batch-normalized advantage from (12), and the clip range ϵ_{clip} is distinct from the stability constant ε_0 in (10)/(12) and from the noise variable ϵ in (9).

Our dual-channel advantage. We instantiate the GDPO pipeline above with $n = 2$. The per-channel advantages $A_i^{(1)}, A_i^{(2)}$ are obtained from (10) with $r_i^{(k)}$ being the unimplemented and unintended reward counts of Eq. (6), the group-level advantage $A_i^{\text{sum}} = w_1 A_i^{(1)} + w_2 A_i^{(2)}$ follows (11) (cf. Eq. (7)), and the batch-normalized advantage $\hat{A}_i$ is given by (12) (cf. Eq. (8)). Plugging $\hat{A}_i$ into (14) yields the final training objective

$$\mathcal{J}_{\text{ours}}(\theta) = \mathbb{E}_{(\mathbf{x}_{\text{src}}, c) \sim \mathcal{D}} \mathbb{E}_{G \sim \pi_{\theta_{\text{old}}}} \mathbb{E}_{i,t} \left[\min(\rho_{i,t}(\theta) \hat{A}_i, \text{clip}(\rho_{i,t}(\theta), 1 - \epsilon_{\text{clip}}, 1 + \epsilon_{\text{clip}}) \hat{A}_i) \right], \quad (15)$$

where the outer expectation is over training pairs from $\mathcal{D}$, the middle expectation over groups sampled from $\pi_{\theta_{\text{old}}}$, and the inner expectation over candidate index i and denoising step t .

C Additional Experimental Results

C.1 Additional Validation

Off-target pixel modification. We randomly selected 600 examples from the evaluation benchmarks. For each source image $\mathbf{x}_{\text{src}}$ and instruction, SAM 2 [Ravi et al., 2024] was used to segment the target object and produce a mask M covering the region expected to change. For edited image $\mathbf{x}_{\text{edit}}$, we measure the mean absolute difference outside that mask:

$$\text{OffTarget-L1} = \frac{\|(1 - M) \odot (\mathbf{x}_{\text{edit}} - \mathbf{x}_{\text{src}})\|_1}{\|1 - M\|_1}. \quad (16)$$

All four methods in Table 6 use the same mask protocol. EditScore(PF, Cons)-GDPO replaces ARRO’s two issue-count rewards with EditScore Prompt-Following and Consistency under fixed GDPO (Table 8). EditScore-Top1-SFT instead fine-tunes on the highest-EditScore candidate in each source–instruction group; it is distinct from Top-10 SFT, which selects the highest-scoring 10% of MinEval candidates (Table 7). OffTarget-L1 also depends on mask quality and image alignment and does not measure semantic correctness.

Table 6: Off-target pixel change on 600 randomly selected evaluation examples. Relative improvement is computed against the base editor. Lower OffTarget-L1 is better.

Method	OffTarget-L1 ↓	Improvement vs. Base
FLUX.1 Kontext-dev (Base)	0.20366	—
EditScore-Top1-SFT	0.19952	2.0%
EditScore(PF, Cons)-GDPO	0.18975	6.8%
ARRO	0.18659	8.4%

Reward construction and offline selection. The raw-audit variant directly uses initial open-ended issue counts, without aggregation or rubric-anchored verification. The Top-10 SFT variants select the highest-reward 10% of MinEval candidates under either ARRO or EditScore reward and use them as targets for LoRA fine-tuning. ARRO-GDPO instead uses within-group relative rewards across candidates. Table 7 reports a separate MinEval-only comparison on the 600-example split: its full-method ES is 5.6690, while the four-benchmark experiment in Table 1 reports 5.73 on MinEval. The four-benchmark mean uses the MinEval result from that experiment. The no-aggregation variant retains verification without consolidating semantically overlapping issues; its four-benchmark means are ES 4.961, HPSv2 0.2101, and ImageReward -0.7672 , compared with 5.880, 0.2220, and -0.5100 for full ARRO. Raw-audit results are only available on MinEval, so we do not present them as a four-benchmark mean.

Table 7: Reward-pipeline and Top-10 SFT controls on the same 600-example MinEval evaluation. All methods use the same EditScore, HPSv2, and ImageReward protocols.

Method	PF $\uparrow$	Cons. $\uparrow$	PQ $\uparrow$	ES $\uparrow$	HPSv2 $\uparrow$	IR $\uparrow$
Base	5.9920	8.1573	6.3393	4.7237	0.2120	-0.7991
Raw audit counts	5.0843	7.0832	4.0630	3.2779	0.1911	-1.0830
EditScore Top-10 SFT	5.7700	8.2827	6.2713	4.5529	0.2091	-0.8467
ARRO Top-10 SFT	5.7893	8.3240	6.3933	4.6289	0.2082	-0.8782
ARRO-GDPO	7.1113	8.1700	6.5693	5.6690	0.2196	-0.6309

EditScore rewards under fixed GDPO. To test whether the gain is merely due to GDPO or reward objectives that resemble EditScore, we replace ARRO’s two verified error-count signals with EditScore Prompt-Following and Consistency scores while retaining the GDPO configuration and training examples. Table 8 shows that ARRO’s structured rewards outperform this control on all four benchmarks. These are evaluation ES values; the EditScore sub-scores used as training rewards in the control are distinct from the reported overall ES.

Table 8: EditScore overall under fixed GDPO training with different reward signals. Average is unweighted across four benchmarks.

Reward	MinEval	MagicBrush	AnyBench	Emu-Edit	Average
EditScore(PF, Cons)	3.566	3.233	4.379	3.932	3.777
ARRO verified issue counts	5.730	5.800	6.070	5.910	5.880

On the three unseen benchmarks alone (MagicBrush, AnyBench, and Emu-Edit), the corresponding ES averages are 3.848 for EditScore(PF, Cons)-GDPO and 5.927 for ARRO. These three-benchmark means exclude MinEval and differ from the four-benchmark averages in Table 8.

Controlled test of intended versus unintended change. In a CLEVR-based [Johnson et al., 2017] diagnostic, each candidate implements all object recolorings specified by its instruction, so the ground-truth unintended-change count is zero. The number of requested recolorings increases from 5 to 50, with five samples per condition. ARRO uses its training-time Audit–Aggregation–Verification procedure; EditScore Consistency uses the public Qwen3-VL-8B evaluator and its official LoRA adapter with three inference runs. The reported means in Table 9 show that the verified ARRO count remains zero across these controlled conditions, while EditScore Consistency varies despite no off-target change. This diagnostic probes the two evaluators’ treatment of intended edits; it does not establish behavior on all natural images.

Table 9: CLEVR diagnostic: every specified modification is implemented and the ground-truth unintended-change count is zero. Each column reports a five-sample mean.

Requested modifications	5	10	15	20	25	30	35	40	45	50
ARRO unintended count	0	0	0	0	0	0	0	0	0	0
EditScore Consistency	9.200	8.333	8.867	8.467	8.733	8.667	8.533	9.240	8.867	9.000

Evaluator scale and reward-generation cost. For a candidate group of size N , the reward pipeline makes N VLM audit calls, one LLM aggregation call, and N VLM verification calls, for $2N + 1$ inference passes. With $N = 16$, Qwen3-VL-30B-A3B-Instruct and Qwen3-30B-A3B-Instruct-2507 were served on separate NVIDIA A100-SXM4-80GB GPUs at maximum concurrency 128. The complete reward-generation stage took 16.06 seconds per group and processed 61,950 tokens (54,720 prompt and 7,230 completion tokens). Candidate-level audit and verification requests can run concurrently; this measured latency is for the stated hardware and server configuration, not an end-to-end training throughput result.

We also tested smaller evaluator pairs on a separate deterministic CLEVR [Johnson et al., 2017] diagnostic: five scenes, each under 0, 1, 2, or 3 visually salient off-target modifications, giving 20 outputs and five samples per ground-truth count. All five requested object changes were implemented. The VLM performs audit and verification, and the paired LLM performs aggregation, with identical images, prompts, decoding settings, and pipeline. Table 10 reports the mean predicted count by ground-truth condition, sample-level MAE, and exact-count accuracy. These results support a cost-accuracy trade-off for this controlled count task, without measuring full editing-model training at each evaluator scale.

Table 10: Evaluator-size diagnostic on 20 CLEVR outputs. GT columns show the average predicted unintended-change count among five examples with that ground-truth count.

VLM + LLM	GT=0	GT=1	GT=2	GT=3	MAE ↓	Exact acc. ↑
Qwen3 4B + 4B	0.80	1.40	1.80	3.40	0.75	0.40
Qwen3 8B + 8B	0.80	1.00	2.00	2.80	0.55	0.50
Qwen3 32B + 30B	0.40	1.20	1.80	3.00	0.30	0.70

Motivation and scope of MinEval. Each MinEval instruction targets one atomic editing objective and allows multiple valid outputs. For each source–instruction pair, grouped candidate edits and separate intended- and unintended-change assessments provide graded supervision without one ground-truth edited image. This structure supports the RL comparison studied here and candidate filtering, as illustrated by the Top-10 SFT controls. Uses for disentangled representation learning remain prospective rather than experimentally established here.

C.2 Human Evaluation

Expert validation of issue judgments. We sampled 60 MinEval groups balanced across addition, removal, replacement, movement, action change, and appearance change. Each group contains a shared source image and instruction with 16 edited candidates, yielding 960 candidate images and 565 final aggregated issues. Candidate identities and presentation order were randomized. Two independent second-year computer-science master’s students, not involved in this work, reviewed the source, instruction, candidate images, raw audit issues, and verified issues without model identities, reward values, or reward-generation procedures. They independently checked whether repeated issues were merged, whether final issues were supported by the source and edited images, and whether an evident failure was missing. Table 11 reports conservative counts jointly confirmed by both annotators; these are issue- and group-level correctness checks, not inter-rater agreement coefficients. Candidate assignment remained a source of error and is not quantified here.

Table 11: Expert validation of the current reward pipeline. Fractions use issues for the first two rows and candidate groups for the third.

Check	No error confirmed by both	Error confirmed by both
Repeated issues merged correctly	560/565 (99.1%)	5/565 (0.9%)
Final issue visually supported	553/565 (97.9%)	12/565 (2.1%)
No evident issue omitted	58/60 (96.7%)	2/60 (3.3%)

Blinded pairwise preference. We sampled 60 MinEval examples balanced across the same six edit types. Ten independent second-year computer-science master’s students, separate from the issue

annotators and not involved in this work, each compared the source image, instruction, and two model outputs for all 60 examples. Model identities and metric scores were hidden; left/right presentation was randomized. Evaluators chose the better edit or a tie. The 600 judgments per comparison in Table 4 share 60 underlying examples, so they should not be interpreted as 600 independent image pairs. The prompt asked for overall editing quality rather than a separate locality-only preference.

C.3 Transfer to OmniGen2

To assess transfer beyond FLUX.1 Kontext-dev, we trained OmniGen2 with the same ARRO Audit–Aggregation–Verification reward construction and rejection-sampling strategy. OmniGen2 used its official training script and recommended configuration with FlowGRPO; thus the generator and RL implementation both differ from the FLUX.1 Kontext-dev/MixGRPO experiment. The three unseen benchmark results in Table 12 consistently improve under this adaptation. The same 60 MinEval examples, evaluators, and blinded protocol used in the pairwise study above give the human result in Table 4. This demonstrates transfer to a second rectified-flow editor, without claiming diffusion-backbone validation.

Table 12: OmniGen2 transfer on three unseen benchmarks. Average is the unweighted mean of benchmark scores.

Benchmark	Model	ES $\uparrow$	HPSv2 $\uparrow$	ImageReward $\uparrow$
MagicBrush	OmniGen2	5.03	0.221	−0.59
	OmniGen2 + ARRO	5.58	0.228	−0.33
AnyBench	OmniGen2	4.71	0.215	−0.68
	OmniGen2 + ARRO	5.22	0.221	−0.58
Emu-Edit	OmniGen2	5.44	0.207	−0.77
	OmniGen2 + ARRO	5.99	0.213	−0.60
Average	OmniGen2	5.06	0.214	−0.68
	OmniGen2 + ARRO	5.59	0.221	−0.50

C.4 EditScore Sub-Scores

The EditScore evaluator [Luo et al., 2025] produces three sub-scores in addition to the Overall score reported in the main paper:

- **EditScore-PF (PF)** — prompt following: did the requested edit happen? Range $[0, 10]$, $\uparrow$.
- **EditScore-Cons (Cons)** — consistency: are non-target regions preserved? Range $[0, 10]$, $\uparrow$.
- **EditScore-PQ (PQ)** — perception quality: is the output clean and artifact-free? Range $[0, 10]$, $\uparrow$.

The Overall score reported in Tables 1 and 3 is a joint judgment combining these three axes. Table 13 reports the per-benchmark PF/Cons/PQ breakdown for all methods, complementing the Overall numbers in Table 1. Two observations relevant to the main-paper discussion: **(i)** ARRO delivers a large gain on prompt-following over the base model ($+0.75$ averaged over the four benchmarks: $6.60 \rightarrow 7.35$), while its Consistency stays essentially flat ($8.36 \rightarrow 8.31$, -0.05 on average), confirming that the Overall improvement is driven by better instruction execution rather than by larger, more disruptive edits. **(ii)** Among the baselines, the largest jumps in Overall are typically explained by PF, not by Cons; methods such as InstructPix2Pix and the MagicBrush model achieve middling Cons by under-editing (low PF), illustrating why the joint Overall metric is the appropriate headline number.

C.5 Per-Benchmark HPSv2 and ImageReward

Table 14 reports the per-benchmark HPSv2 and ImageReward values that complement the per-benchmark EditScore Overall in Table 1. ARRO is highest on HPSv2 on every benchmark (tied with InstructPix2Pix on MinEval) and highest on ImageReward on every benchmark.

Table 13: Per-benchmark EDITSCORE sub-scores (PF, Cons, PQ) for all methods, complementing the Overall numbers in Table 1. All sub-scores lie in $[0, 10]$, $\uparrow$. **Avg** is the unweighted mean across the four benchmarks. Best per column in **bold**.

Method	MinEval			MagicBrush			AnyBench			Emu-Edit			Avg		
	PF	Cons	PQ	PF	Cons	PQ	PF	Cons	PQ	PF	Cons	PQ	PF	Cons	PQ
FLUX.1 Kontext-dev (Base)	6.00	8.12	6.39	6.74	8.50	6.57	6.64	8.48	6.70	7.03	8.33	6.15	6.60	8.36	6.45
InstructPix2Pix	2.31	6.13	6.24	3.24	6.65	6.20	3.34	6.98	6.39	3.34	6.60	6.06	3.06	6.59	6.22
MagicBrush model	4.05	5.94	4.82	5.63	7.22	5.29	4.44	5.63	4.37	4.73	6.13	5.02	4.71	6.23	4.87
OmniGen2	6.03	8.77	7.13	6.11	9.04	7.22	6.03	8.46	6.76	6.78	8.54	6.49	6.24	8.70	6.90
UniWorld-v1	5.08	5.56	3.95	5.12	6.18	3.97	5.27	5.97	4.67	5.14	5.64	3.80	5.15	5.84	4.10
ICEdit	5.50	8.78	7.56	6.50	8.74	7.33	5.05	8.15	7.12	6.21	8.37	6.93	5.81	8.51	7.24
BAGEL	7.81	8.19	6.04	8.13	8.56	6.18	6.80	8.01	5.95	7.91	7.78	5.79	7.67	8.14	5.99
ARRO (ours)	7.14	8.19	6.60	7.48	8.31	6.56	7.28	8.51	6.90	7.49	8.23	6.45	7.35	8.31	6.63

Table 14: Per-benchmark HPSv2 (HPS) and ImageReward (IR) for all methods. Best per column in **bold**. EditScore Overall per benchmark is in Table 1.

Method	MinEval		MagicBrush		AnyBench		Emu-Edit	
	HPS	IR	HPS	IR	HPS	IR	HPS	IR
FLUX.1 Kontext-dev (Base)	0.212	-0.80	0.224	-0.51	0.215	-0.58	0.208	-0.78
InstructPix2Pix	0.220	-0.67	0.213	-0.96	0.212	-0.85	0.199	-1.08
MagicBrush model	0.211	-0.84	0.211	-0.86	0.205	-0.88	0.199	-1.00
OmniGen2	0.211	-0.83	0.221	-0.59	0.215	-0.68	0.207	-0.77
UniWorld-v1	0.203	-0.92	0.206	-0.90	0.204	-0.87	0.196	-0.99
ICEdit	0.209	-0.82	0.212	-0.70	0.208	-0.81	0.199	-0.89
BAGEL	0.208	-0.84	0.214	-0.59	0.208	-0.75	0.207	-0.72
ARRO (ours)	0.220	-0.63	0.233	-0.29	0.221	-0.49	0.214	-0.62

C.6 Per-Edit-Type Breakdown on AnyBench

Tables 15 and 16 report the per-edit-type HPSv2 and ImageReward breakdown on AnyBench (50 prompts per edit type), complementing the EditScore table in the main paper (Table 2). The methods, edit-type columns, and best/second-best (*) marking convention all match Table 2.

D Training Settings

We list the key hyperparameters used to train ARRO in Tables 17 and 18. All FLUX.1 Kontext-dev runs share this configuration; ablations in Sec. 4.4 change only their specified reward components.

Table 15: Per-edit-type HPSv2 on AnyBench (50 prompts per edit type), complementing Table 2 in the main paper. Edit-type columns and method order match Table 2. Higher is better. The best score in each column is shown in **bold**, and the second-best score is marked with *.

Method	Edit type																	
	act.	add	app.	bg.	color	count	impl.	mat.	move	outp.	rel.	rem.	repl.	resize	rot.	style	text	tone
Base	0.266*	0.266*	0.245*	0.210	0.223	0.177	0.240	0.221	0.210	0.191*	0.215	0.169	0.234*	0.221	0.183*	0.191	0.233	0.177
ARRO	0.269	0.268	0.255	0.218	0.234	0.182	0.251	0.227	0.219*	0.193	0.219	0.165	0.241	0.226*	0.184	0.202	0.231	0.187
IP2P	0.247	0.229	0.222	0.198	0.230	0.203	0.226	0.230*	0.221	0.170	0.218*	0.207	0.222	0.230	0.173	0.190	0.227	0.166
MagicBr.	0.233	0.240	0.228	0.222*	0.221	0.181	0.216	0.217	0.211	0.165	0.203	0.193*	0.217	0.216	0.161	0.175	0.208	0.190*
OmniGen2	0.261	0.252	0.242	0.222	0.235*	0.172	0.247*	0.234	0.187	0.188	0.205	0.173	0.229	0.217	0.176	0.200	0.236	0.190
UniWorld-v1	0.250	0.249	0.226	0.196	0.229	0.168	0.232	0.227	0.196	0.175	0.204	0.168	0.220	0.198	0.175	0.201*	0.195	0.167
ICEdit	0.233	0.241	0.233	0.183	0.237	0.188*	0.213	0.227	0.203	0.180	0.210	0.192	0.225	0.203	0.172	0.194	0.229	0.184
BAGEL	0.266	0.260	0.232	0.203	0.226	0.168	0.247	0.220	0.193	0.175	0.193	0.173	0.214	0.204	0.175	0.193	0.234*	0.176

Table 16: Per-edit-type IMAGEREWARD on AnyBench (50 prompts per edit type), complementing Table 2 in the main paper. Edit-type columns and method order match Table 2. Higher is better. The best score in each column is shown in **bold**, and the second-best score is marked with *.

Method	Edit type																	
	act.	add	app.	bg.	color	count	impl.	mat.	move	outp.	rel.	rem.	repl.	resize	rot.	style	text	tone
Base	0.42*	0.79	0.27*	-0.81	-0.15	-1.33	0.05	-0.79	-0.46	-1.12	-0.49	-1.76	-0.66	-0.44	-1.57*	-1.15	-0.09	-1.18
ARRO	0.56	0.78*	0.42	-0.59	0.08*	-1.21	0.32	-0.54*	-0.35	-0.99*	-0.40	-1.80	-0.76*	-0.47*	-1.55	-1.02	-0.21	-1.07
IP2P	-0.43	-0.61	-0.86	-1.16	-0.50	-0.81	-0.53	-0.74	-0.38*	-1.44	-0.44*	-1.10	-1.18	-0.55	-1.73	-1.19	0.02	-1.61
MagicBr.	-0.36	-0.14	-0.49	-0.58*	-0.60	-1.22	-0.55	-1.01	-0.67	-1.43	-0.63	-1.34*	-1.12	-0.81	-1.88	-1.27	-0.62	-1.15*
OmniGen2	0.05	0.34	0.02	-0.54	-0.02	-1.48	-0.05	-0.51	-1.05	-0.99	-0.61	-1.91	-0.94	-0.59	-1.70	-1.06*	-0.07*	-1.15*
UniWorld-v1	0.14	0.25	-0.35	-0.96	-0.26	-1.48	-0.46	-0.87	-0.91	-1.28	-0.66	-1.78	-0.90	-0.98	-1.74	-1.07	-0.76	-1.57
ICEdit	-0.68	0.06	-0.26	-1.31	0.10	-1.09*	-0.63	-0.60	-0.81	-1.24	-0.55	-1.47	-0.91	-0.84	-1.68	-1.23	-0.23	-1.18
BAGEL	0.23	0.72	-0.16	-0.89	-0.16	-1.44	0.20*	-0.63	-0.84	-1.29	-0.86	-1.84	-1.16	-0.87	-1.75	-1.33	-0.11	-1.35

Table 17: Key hyperparameters of ARRO fine-tuning on FLUX.1 Kontext-dev.

Parameter	Setting
<i>Optimizer</i>	
Optimizer	AdamW
Learning rate	1×10^{-5}
Weight decay	1×10^{-4}
Gradient-norm clip	1.0
PPO ratio clip	5×10^{-3}
Precision	bf16
<i>Sampler (training)</i>	
Resolution	720×720
Denoising steps T	25
Flow-match shift	3
Stochasticity η	0.7
Group size N (candidates per prompt)	16
<i>MixGRPO</i>	
Sliding window	first 60% of schedule
iters_per_group	20
group_size (sliding-window size, in steps)	4
Progressive overlap	enabled

Table 18: Reward, compute, and inference settings for FLUX.1 Kontext-dev experiments.

Parameter	Setting
<i>Reward</i>	
Channel-1 weight w_1	0.3
Channel-2 weight w_2	0.7
Auditor VLM	Qwen3-VL-30B-A3B-Instruct [Qwen Team, 2025]
Aggregator LLM	Qwen3-30B-A3B-Instruct-2507 [Qwen Team, 2025]
<i>Compute</i>	
Hardware	8 × A100-80GB (single node)
Per-GPU prompts/step	1
Gradient accumulation	4
Effective batch	32 prompts/step
Total optimizer steps	230
Wall-clock	≈300 GPU-hr
<i>Sampler (inference)</i>	
Resolution	720 × 720
Denoising steps T	25
Guidance scale	3.5
Stochasticity η	0
Seed	42 (deterministic)

E Training Data Construction

This appendix expands the three-stage data pipeline summarized in Sec. 4.1. Stage (i) synthesizes atomic instructions, stage (ii) filters them through a structured VLM audit, and stage (iii) executes them with the un-tuned base editor and rejects instructions whose candidate score distributions are degenerate.

E.1 Atomic Edit Types

Every instruction is constrained to one of six atomic edit types, each with a fixed surface pattern and a closed list of allowed verbs:

- **add** — “*add/insert/put/place* <object> *to/at* <location>”.
- **remove** — “*remove/delete/erase* <object>”.
- **move** — “*move/relocate* <object> *from* <src> *to* <dst>”.
- **replace** — “*replace/substitute/swap* <old object> *with* <new object>”.
- **appearance_change** — “*change/make/paint* <object>’s <attribute> *to* <value>” (color, texture, material).
- **action_change** — “*make/have* <animate object> <action>” (action must be species-appropriate).

Instructions whose leading verb conflicts with the declared edit type (e.g., a **remove**-style verb under **add**) are rejected at parse time before any VLM call.

E.2 Instruction Generation and Validation

For each source image x_{src} and edit type t , the VLM is invoked in one of two modes. When a draft instruction c_0 is available (e.g., from a prior pass), the *validation* prompt P_{val} asks the VLM to audit c_0 against a fixed checklist and, when fixable, return a corrected instruction. When no draft is available, or after 5 failed validation rounds, the *generation* prompt P_{gen} asks the VLM to propose a fresh instruction from scratch and self-audit it under the same checklist. To improve diversity, the generation call is launched in parallel with 5 different temperatures $\{0.6, 0.7, 0.8, 0.9, 1.0\}$ and the first candidate that passes the checklist is kept. Both prompts share the same JSON output schema; the validation variant adds an explicit instruction field for the corrected wording, the generation

variant a `generated_instruction` field. We show the validation variant below; the generation variant is identical except that the `<instruction_to_validate>` block is removed and Step 0 is replaced by “select a single, high-confidence target object”.

$P_{\text{val}}(x_{\text{src}}, c_0, t)$ (instruction-filtering prompt)

Role: You are validating an image editing instruction. Decide whether the instruction is accurate, complete, and feasible for the given source image.

Inputs:

- **Source image:** $\{x_{\text{src}}\}$
- **Instruction to validate:** $\{c_0\}$
- **Edit type:** $\{t\}$ (one of add, remove, move, replace, appearance_change, action_change).
- **Required pattern** for t : as defined in App. E.1.

Step 1: Single-object check. Does the instruction target exactly one object? Reject phrasings such as “and”, “both”, “all”, or plural targets.

Step 2: Object verification. Look at the source image and decide whether the referenced object actually exists and is correctly identified (i.e., not confused with a similar-looking object).

Step 3: Completeness check. Verify that all slots required by the edit type are filled (e.g., replace requires both old and new object; move requires both source and target locations; add requires both object and location).

Step 4: State-change check. Compare the current state of the target with the state requested by the instruction. If they already match, the edit is trivial and must be rejected.

Step 5: Feasibility check. Decide whether the requested change is physically plausible; for action_change, additionally check that the action is appropriate for the species of the target.

Output format. Return a single JSON object and nothing else:

```
{
  "parsed_instruction": {
    "action": "<verb>", "target": "<single object>", "details": "<...>"
  },
  "single_object_check": {
    "targets_single_object": true|false,
    "violation_found": "<...>"
  },
  "object_verification": {
    "object_exists": true|false,
    "object_correctly_identified": true|false,
    "what_i_actually_see": "<...>"
  },
  "completeness_check": {
    "all_required_fields_present": true|false,
    "missing_fields": ["<...>"]
  },
  "state_verification": {
    "current_state": "<...>",
    "requested_state": "<...>",
    "states_are_different": true|false
  },
  "feasibility_check": {
    "physically_possible": true|false,
    "species_appropriate": true|false|null,
    "issue": "<... or 'none'>"
  },
  "overall_result": {
    "feasible": true|false,
    "instruction": "<original if valid, corrected if fixable, else empty>",
    "failed_checks": ["<names of any failed checks>"],
    "reason": "<brief summary>"
  }
}
```

An instruction is accepted only if every check returns true and the instruction field, after pipeline-level verb-vs-edit-type re-check, is non-empty. Otherwise the failed-check names are fed back to the VLM as a recheck context for up to 5 rounds, after which the pipeline falls back to parallel generation as described above. Instructions that never pass either route are discarded.

E.3 Per-edit-type guidance injected into P_{val} and P_{gen}

Both P_{val} and P_{gen} inject two type-specific blocks under the placeholder “*Required pattern for t* ”: a **type-specific guidance** block (what makes a valid instruction of this type) and an **evaluation-alignment** block (which verifiable criteria the downstream agentic reward model will check). The six blocks below are summaries of the strings we actually inject for each t .

Edit type add

Pattern. Add [specific object with attributes] to/in [precise location]. **Allowed verbs:** *add, put, place, insert*.

Type-specific guidance. Introduce a *new* object that does not yet exist in the image. Specify (i) the object: type, color, size (e.g., “a large red balloon”); (ii) a precise location with confirmed empty space. Reject if the candidate location lacks empty space sufficient for the object’s size, since adding the object would otherwise displace existing content and create side effects.

Evaluation alignment. The downstream reward model checks: *presence* (the new object appears), *quantity* (count matches; always state quantity explicitly), *position* (at the specified location), *attributes* (color/size/type match), and *addition $\neq$ replacement* (existing objects must remain). Use *add/place*, never *replace/swap*. Prefer empty-space targets so the evaluator can confirm no displacement.

Edit type remove

Pattern. Remove the [color/feature] [object] [position disambiguator]. **Allowed verbs:** *remove, delete, erase*.

Type-specific guidance. Remove an object that clearly exists and can be uniquely identified. Confirm the object is visible, add a disambiguator when multiple similar objects exist, and verify the identification is correct (not a similar-looking object, e.g., sheep mistaken for bull).

Evaluation alignment. The downstream reward model checks: *true removal* (the object must be completely invisible; blurring, painting-over, or covering counts as failure) and *disambiguation* (the evaluator must be able to locate the exact target). Use *remove/delete/erase*, never *hide/cover/obscure* (which imply the object is still present). Add enough disambiguators (color, position, size) when multiple similar objects exist.

Edit type move

Pattern. Move the [object] from [current position] to [different target position]. **Allowed verbs:** *move, shift, relocate*.

Type-specific guidance. Relocate an existing object to a clearly different position. Verify (i) current position (where the object *is now*) and (ii) target position (where it should *go*, must be different from current). Use canonical position equivalences (LEFT=“left side”, CENTER=“middle”, etc.) when checking that source $\neq$ destination.

Evaluation alignment. The downstream reward model checks: *destination* (object present at the new position), *source clear* (gone from the original; no residual or ghost), *no duplication* (object must not appear at both positions), *identity* (same object preserved). Always specify both “from [source]” and “to [destination]” so the evaluator can verify both positions; vague phrasing such as “move the cup over” fails because the evaluator cannot confirm source-clear.

Edit type replace

Pattern. Replace the [existing object] with [specific new object]. **Allowed verbs:** *replace, substitute, swap*.

Type-specific guidance. Both the old and the new object must be specified explicitly. The old object must be visible in the source image; the new object should be similar in size/shape so it can fit naturally at the same location without major adjustments to the surrounding scene.

Evaluation alignment. The downstream reward model checks: *A gone* (old object completely absent), *B present* (new object appears at A’s original location), and *no coexistence* (A and B must not both be visible; coexistence = failure). Name the old object precisely so the evaluator can confirm absence. Use “replace A with B”, never just “add B” (which implies A stays).

Edit type appearance_change

Pattern. Change the [object]’s [attribute] from [current value] to [new value] (color, material, texture). **Allowed verbs:** *change, modify, turn, make*.

Type-specific guidance. Modify a single appearance attribute of an existing object. Verify (i) the object exists, (ii) its current attribute value is identified accurately, and (iii) the target value is different from the current one. Reject when current value already equals target (e.g., “change the stop sign to red”), or when the named attribute does not exist on the object (e.g., “change the saddle” on an unsaddled horse).

Evaluation alignment. The downstream reward model checks: *identity preserved* (object keeps its shape, position, and category), *attribute matches* (the specified attribute reaches the target value), and *only the specified attribute changes*. Be specific about which attribute is being changed (“color from red to blue”, not “make it different”); vague instructions cause false failures because the evaluator cannot determine the intended boundary.

Edit type action_change

Pattern. Make the [specific subject] [new action] instead of [current action]. **Allowed verbs:** *make, have, change*.

Type-specific guidance. Change a subject’s pose, action, or expression to something different and *species-appropriate*. Use species-appropriate action vocabularies (humans: sit/stand/walk/wave/lie down; quadrupeds: lie down/walk/turn head/graze, but not “stand up” since they are already standing on four legs; birds: perch/fly/spread wings/hop). Reject when current action equals target (e.g., “make the person sit down” on someone already sitting).

Evaluation alignment. The downstream reward model checks: *correct subject* (the right person/animal performs the action; disambiguate when multiple subjects exist), *action visible* (target action is clearly visible in a static frame), *subject identity preserved*. Choose actions that are visually verifiable in a single frame; avoid temporally extended actions like “start walking” or “begin to speak” that require motion to verify.

E.4 Edit-Level Rejection Sampling

The instructions that survive App. E.2 are then filtered at the edit level. For every $(\mathbf{x}_{\text{src}}, c, t)$ triplet we sample a group of $N=16$ candidate edits $\mathcal{G}=\{\mathbf{x}_i\}_{i=1}^N$ from the un-tuned base editor (FLUX.1 Kontext-dev) at the training resolution of 720×720 , with $T=25$ denoising steps and i.i.d. noise seeds; this matches the GDPO group used during RL fine-tuning (Sec. 4.2). Each candidate is then scored by the same three-step agentic reward pipeline used at training time: audit, group-level rubric aggregation, and rubric-anchored verification (Eqs. (3)–(6)), yielding the same two reward channels we optimize against:

$$r_i^{(1)} = -|\mathcal{U}_i|, \quad r_i^{(2)} = -|\mathcal{O}_i|,$$

that is, the negative counts of verified unimplemented and unintended issues for candidate $\mathbf{x}_i$.

To align with the GDPO advantage estimator, which is identically zero on a group whose rewards are constant, we filter on the variation of these two channels within $\mathcal{G}$. Concretely, we run a two-pass filter:

- *Pass 1 (unimplemented channel)*. Drop the group if more than half of its candidates have $|\mathcal{U}_i|=0$, i.e., if $\#\{i : |\mathcal{U}_i|=0\} > N/2$. When a majority of candidates already realize the instruction c in full, the channel-1 reward $r_i^{(1)}$ is dominated by a single mode at 0, the within-group variance of the unimplemented signal collapses, and GDPO receives little advantage signal from this channel. Such groups are dropped regardless of their channel-2 distribution.
- *Pass 2 (unintended channel)*. Among groups that pass Pass 1, drop those for which $|\mathcal{O}_i|$ is also constant across i , or for which $\text{std}(|\mathcal{O}_{1:N}|) < \sigma_{\min}$, or for which the per-group mean of $|\mathcal{O}_i|$ exceeds a maximum tolerated count $\mu_{\max}$. The first two conditions ensure the locality channel actually carries a within-group signal; the last condition discards prompts on which the base editor changes so much of the source image that the auditor would have to reason over a long, heterogeneous list of differences, which keeps per-item judgments focused and low-variance.

For the runs reported in Sec. 4 we use $\sigma_{\min}=0$ and $\mu_{\max}=999$, so that within Pass 2 only the constant-across- i test on $|\mathcal{O}_i|$ is active (the std and mean tests are inert at these settings). In this default configuration the filter discards two kinds of groups: those whose unimplemented channel is dominated by zeros (Pass 1, majority-success), and those whose unintended channel is constant across the group (Pass 2, locality-channel constancy). Both are degenerate cases for the group-relative advantage estimator: in the first, the efficacy channel carries little ranking signal because most candidates already realize c in full; in the second, the locality channel literally has zero within-group variance and contributes a zero advantage. Tightening $\sigma_{\min}$ or $\mu_{\max}$ yields the more aggressive variants used in our preliminary tuning runs but did not change the qualitative ranking. The retained triplets form the $\sim 10\text{K}$ atomic-edit training set used for RL fine-tuning.

F Prompt Templates

We instantiate the three model invocations in Sec. 3 with the following prompt templates. Each template is a fixed string with placeholders (in `{curly braces}`) for inputs. All templates instruct the model to return a JSON object with a fixed schema, which we parse to obtain the required outputs in Eqs. (3), (4), and (5).

The audit prompt P_{audit} and the verification prompt P_{verify} are designed to share the same reasoning structure: both ask the VLM to parse the instruction, compare the source and edited images, and identify changes belonging to the two failure modes. They differ only in the output mode. The audit is open-ended, asking the VLM to generate the lists $\tilde{\mathcal{U}}_i$ and $\tilde{\mathcal{O}}_i$ from scratch. The verification is closed-form, asking the VLM to select, from each fixed group rubric, the items present in x_i . This shared structure ensures the two stages apply the same notion of "unimplemented" and "unintended" change, and the rubric refinement loop only varies what the VLM is asked to *produce*, not how it is asked to *look*.

F.1 Audit prompt P_{audit}

The audit prompt is given to the VLM Φ_V together with the source image x_{src} , the candidate edit x_i , and the instruction c , and produces the raw lists $(\tilde{\mathcal{U}}_i, \tilde{\mathcal{O}}_i)$.

$P_{\text{audit}}(x_{\text{src}}, c, x_i)$

Role: You are a careful visual auditor. You will be shown two images and a natural-language editing instruction. Your task is to identify two kinds of discrepancies between the instruction and what the edited image actually shows.

Inputs:

- **Source image:** $\{x_{\text{src}}\}$
- **Edited image:** $\{x_i\}$
- **Editing instruction:** $\{c\}$

Step 1: Parse the instruction. Identify what the instruction explicitly asks to modify, what the resulting state should be, and what should remain unchanged.

Step 2: Compare the two images. Examine the source and edited image side by side and note all visible differences. For each difference, decide whether it falls inside or outside the scope of the instruction.

Step 3: Generate two lists.

1. **Unimplemented changes ($\tilde{\mathcal{U}}_i$):** aspects of the instruction that the edited image fails to realize, including complete omission, partial execution, or incorrect application. Return an empty list if the instruction is fully realized.
2. **Unintended changes ($\tilde{\mathcal{O}}_i$):** visible modifications that fall outside the scope of the instruction. Include only changes you can actually see; do not speculate. Return an empty list if no such changes are present.

Each entry must be a short, self-contained, visually verifiable description of a single change. Do not assign numerical scores. Do not summarize.

Output format. Return a single JSON object and nothing else:

```
{
  "reasoning": "<your step-1 and step-2 analysis>",
  "unimplemented_changes": [
    "<short description>", ...
  ],
  "unintended_changes": [
    "<short description>", ...
  ]
}
```

F.2 Aggregation prompt P_{agg}

The aggregation prompt is invoked once per candidate group. It receives the two audit collections in separate fields, applies the instruction filter only to unintended changes, and returns both canonical rubrics in one response.

$P_{\text{agg}}(c, \tilde{\mathcal{U}}_{1:N}, \tilde{\mathcal{O}}_{1:N})$

Role: You are a rubric editor. Given an editing instruction and two collections of free-form issues identified across N candidate edits, construct separate, deduplicated checklists for unimplemented and unintended changes. Keep the two failure modes distinct.

Inputs:

- **Editing instruction:** $\{c\}$
- **Raw unimplemented issues:** candidate-indexed lists $\{\tilde{\mathcal{U}}_i\}_{i=1}^N$.
- **Raw unintended issues:** candidate-indexed lists $\{\tilde{\mathcal{O}}_i\}_{i=1}^N$.

Step 1: Filter against the instruction. From the unintended collection only, remove descriptions of changes explicitly requested or directly implied by c . Do not apply this filter to unimplemented issues, which describe missing aspects of c .

Step 2: Consolidate each collection separately. Merge descriptions of the same underlying issue, split descriptions that conflate distinct issues, normalize each entry to one atomic visually checkable failure, and drop vague, speculative, or non-visual entries. For example, “the bow was not removed” and “the bow is still visible” form one unimplemented issue; “the wheels became silver” and “the rims changed color” form one unintended issue when they describe the same change.

Step 3: Produce two rubrics. Write each issue in neutral form so that it can be checked against any candidate. Assign IDs within each rubric. Do not merge an unimplemented issue with an unintended issue.

Output format. Return a single JSON object and nothing else:

```
{
  "reasoning": "<filtering and consolidation rationale>",
  "unimplemented_rubric": [
    {"id": 1, "issue": "<canonical unimplemented issue>"}, ...
  ]
}
```

```

],
"unintended_rubric": [
  {"id": 1, "issue": "<canonical unintended issue>"}, ...
]
}

```

The two returned arrays define $\mathcal{R}_G^U$ and $\mathcal{R}_G^O$, respectively.

F.3 Verification prompt P_{verify}

The verification prompt is invoked once per candidate. It receives both group rubrics and returns separate Boolean verdicts for their issues in one response. As in the audit prompt, the VLM first parses the instruction and compares the source and edited images; verification then selects from the fixed rubrics rather than generating new issues.

$P_{\text{verify}}(x_{\text{src}}, c, x_i, \mathcal{R}_G^U, \mathcal{R}_G^O)$

Role: You are a strict visual verifier. Given a source image, an editing instruction, an edited image, and two checklists of canonical issues, decide which issues are visibly present. Keep unimplemented and unintended changes separate.

Inputs:

- **Source image:** $\{x_{\text{src}}\}$
- **Edited image:** $\{x_i\}$
- **Editing instruction:** $\{c\}$
- **Unimplemented checklist:** $\mathcal{R}_G^U$, with issue IDs and descriptions.
- **Unintended checklist:** $\mathcal{R}_G^O$, with issue IDs and descriptions.

Step 1: Parse the instruction. Identify what c asks to modify, the required result, and what should remain unchanged.

Step 2: Compare the images. Examine the source and edited image side by side. Determine which requested aspects were realized and which visible differences fall outside the scope of c .

Step 3: Select from both checklists. For each unimplemented item, mark `true` only if the named aspect of c remains absent, partial, or incorrect. For each unintended item, mark `true` only if that off-target change is visible. When in doubt, mark `false`. Do not add issues or alter their wording.

Output format. Return a single JSON object and nothing else:

```

{
  "reasoning": "<instruction and image comparison>",
  "unimplemented_verdicts": [
    {"id": 1, "present": true}, ...
  ],
  "unintended_verdicts": [
    {"id": 1, "present": false}, ...
  ]
}

```

The issues marked `true` in the two verdict arrays form $\mathcal{U}_i \subseteq \mathcal{R}_G^U$ and $\mathcal{O}_i \subseteq \mathcal{R}_G^O$. Consequently, one candidate group needs N audit calls, one aggregation call, and N verification calls, for $2N + 1$ reward-generation calls.