EDITJEV: SYSTEM ONE ROUTING FOR IMAGE EDITING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Instruction-based image editing models exhibit pronounced sample-specific performance variation: different models can produce substantially different results for the same source image and editing instruction, and the best-performing model often varies across requests. These models also differ substantially in inference latency, making fixed-model deployment inefficient. We therefore study image editing model routing, which dynamically selects an editor from a candidate pool given an input image and editing instruction. We introduce EditJev, a System One routing model that directly predicts sample-level preference scores over candidate image editing models in a single forward pass. A lightweight preference head on a vision-language backbone produces a fixed-dimensional score vector without autoregressive model-name generation or response parsing. To train EditJev, we construct pairwise preferences from offline candidate outputs and optimize a pairwise ranking objective to learn relative model suitability. At inference time, EditJev combines the predicted preference scores with model-level speed scores to balance editing quality and generation latency. Experiments on ImgEdit and GEdit-En show that EditJev improves aggregate quality over the strongest fixed editor, from 4.540 to 4.592 and from 8.194 to 8.255, respectively, while reducing estimated end-to-end latency by 60.8% and 31.7%. These results demonstrate the effectiveness of direct, structured model routing for efficient image editing.

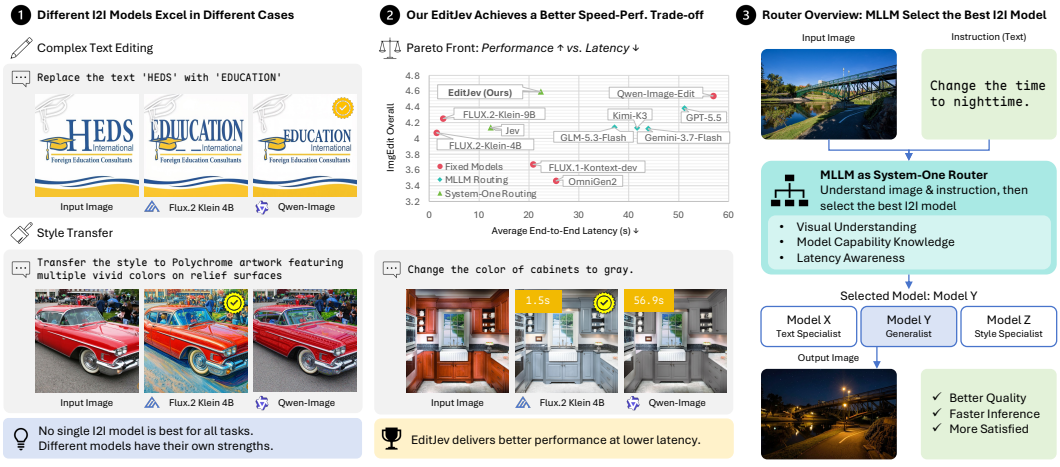

Figure 1: **Left:** Different image editing models exhibit complementary strengths: the optimal model varies across image-instruction pairs, and no single model consistently performs best for all editing requests. **Middle:** This observation motivates model routing, where EditJev achieves a better quality–latency trade-off by dynamically selecting suitable models instead of relying on a fixed model. **Right:** Given an input image and instruction, EditJev uses a multimodal router to understand the request, estimate model suitability, and select the model that provides better editing quality with lower inference latency.

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

1 INTRODUCTION

Instruction-based image editing enables users to modify an image through natural-language commands. A single instruction can specify operations ranging from local object insertion, removal, and attribute adjustment to global style transfer and illumination changes (Brooks et al., 2023; Zhang et al., 2023; Sheynin et al., 2024; Huang et al., 2024). Advances in editing architectures, training data, and model scale have rapidly expanded both the capability and the diversity of available models (Wu et al., 2026; Black Forest Labs et al., 2025; Wu et al., 2025; Cao et al., 2025).

Yet these models are not interchangeable. Differences in architecture, training strategies, and data produce complementary behavior across editing tasks (Zang et al., 2026; Hertz et al., 2022; Huang et al., 2025). A model that performs creative global restyling may alter fine details or spatial structure, whereas a model specialized in precise local manipulation may be less effective at globally consistent transformations (Chen et al., 2025). Their computational profiles are similarly heterogeneous: larger and slower models can be more capable, but they also incur substantially greater memory use and inference latency (Wu et al., 2025; Cao et al., 2025). Thus, model quality and cost depend not only on the model itself but also on the particular image-instruction pair.

This heterogeneity creates a practical deployment dilemma. As illustrated in the left panel of Fig. 1, different image editing models exhibit complementary strengths across image-instruction pairs (Team et al., 2026; Ye et al., 2026). Sending every request to a single default model, typically the largest or most capable overall, can waste computation on easy inputs while overlooking alternatives better suited to particular requests (Zhang et al., 2026; Jiang et al., 2026b; Feng et al., 2026; Chen et al., 2024). Yet manually selecting an editor for each request is impractical, as it requires substantial human effort and detailed knowledge of the candidate models.

Motivated by this observation, we formulate the problem as image editing model routing: given an image, a natural-language instruction, and a set of candidate editors, select a suitable editor before generation while accounting for editing quality and inference cost. The central challenge is that the quality each candidate would achieve is unknown at the time of selection. A router must infer the most suitable editor directly from the input image-instruction pair.

To address this challenge, we introduce EditJev, a System One routing model that learns to predict sample-level preferences over candidate image editing models to guide editor selection. Following the System One principle of efficient decision making through structured prediction (TypeSafe AI, 2026), EditJev represents these preferences as a compact, fixed-dimensional score vector. As illustrated in the right panel of Fig. 1, EditJev equips a vision-language backbone with a lightweight preference head to produce a fixed-dimensional score vector in a single forward pass, requiring only 94 ms for routing inference.

We construct training supervision by evaluating editing outputs generated offline by all candidate models using a multimodal judge. We convert these evaluations into pairwise preferences and train EditJev with a pairwise ranking objective, enabling it to learn relative model suitability for each image–instruction pair. At inference time, as illustrated in the middle panel of Fig. 1, EditJev combines its predicted preference scores with model-level speed scores to select an editor according to the desired balance between editing quality and generation latency.

Our contributions are summarized as follows:

❑ **Problem formulation.** To the best of our knowledge, we are the first to formulate sample-level routing for instruction-based image editing, where a router selects among candidate editing models by jointly considering editing quality and inference latency.

❑ **System One routing framework.** We develop EditJev, a System One routing model that directly predicts preference scores over candidate editors in a single forward pass. Gap-filtered pairwise supervision enables the model to learn the relative suitability of candidate editors and supports latency-aware model selection at inference time.

❑ **Improved quality–latency trade-off.** We evaluate EditJev on ImgEdit and GEdit-En, where it improves aggregate quality over the strongest fixed editor while reducing estimated latency, including routing overhead, by 60.8% and 31.7%, respectively.

2 Related Work

2.1 Instruction-Based Image Editing

Instruction-based image editing modifies an input image according to a natural-language instruction. Prior work has advanced this task through image-instruction datasets (Brooks et al., 2023; Zhang et al., 2023; Hui et al., 2024; Zhao et al., 2024) and dedicated editing frameworks (Sheynin et al., 2024; Huang et al., 2024; Li et al., 2025), supporting diverse operations such as object insertion and removal, attribute modification, and style transformation (Yu et al., 2025; Liu et al., 2025; Xu et al., 2025). Recent methods further improve editing quality by scaling model capacity, training data, and compute (Wu et al., 2026; Black Forest Labs et al., 2025; Wu et al., 2025; Cao et al., 2025; Team et al., 2025; 2026). For example, Qwen-Image-Edit has 20 billion parameters, while HunyuanImage-3.0-Instruct has 80 billion parameters in total (Wu et al., 2025; Cao et al., 2025). Such scaling, however, increases inference latency and deployment cost. Since editing requests vary in difficulty, lightweight models may suffice for some inputs, whereas others require stronger models, motivating dynamic routing to balance editing quality and computational cost.

2.2 Model Routing

Model routing dynamically selects a model from a candidate pool for each input, typically balancing output quality and inference cost. It has been extensively studied for large language models(LLMs) as a strategy for cost-aware serving (Hu et al., 2024; Lai et al., 2026). Existing LLM routers select models by estimating query difficulty, model capability, or expected quality-cost tradeoffs (Ong et al., 2025; Ding et al., 2024; Feng et al., 2025). Model routing has also been extended to text-to-image (T2I) generation (Xin et al., 2025; Jiang et al., 2026a). However, model routing for instruction-based image editing remains underexplored.

Recently, System One models have explored direct structured decision making, where models output compact predictions that can be consumed by downstream systems without additional parsing (TypeSafe AI, 2026). This design principle is particularly relevant to routing problems, where the objective is to select among a fixed candidate set rather than generate open-ended responses. In this work, we introduce EditJev, a direct preference prediction framework that extends model routing to image editing, enabling instance-level selection among editing models.

3 Method

3.1 Problem Formulation

Setting. Let $\mathcal{X}$ denote the space of source images and $\mathcal{C}$ denote the space of editing instructions. We consider a fixed ordered candidate list of image editing models $\mathcal{M} = (M_1, \ldots, M_K)$, where K is the number of candidate models and $[K] = \{1, \ldots, K\}$ denotes the candidate-model index set. Given an editing request $(x, c) \in \mathcal{X} \times \mathcal{C}$, a routing policy $\pi : \mathcal{X} \times \mathcal{C} \to [K]$ selects the index of one candidate model for execution:

$$\hat{i} = \pi(x, c), \qquad \hat{M} = M_{\hat{i}}. \tag{1}$$

For each candidate model M_i, let $Q_i(x, c)$ denote the final quality score of its edited result for request (x, c), and let t_i denote its mean end-to-end generation latency. A desirable routing policy should select a model that achieves a favorable quality–latency trade-off, *i.e.*, high editing quality with low inference latency.

Challenge. At inference time, however, $Q_i(x, c)$ is unknown before candidate model M_i is executed. The key challenge is to learn an efficient routing policy that selects a suitable candidate model using only the source image and editing instruction, without observing any generated outputs.

3.2 System One Routing

Overview. We formulate image editing routing as a System One decision problem, where the router directly predicts numerical preference scores over candidate models rather than autoregressively generating a model choice. As illustrated in Figure 2, EditJev consists of offline preference-based

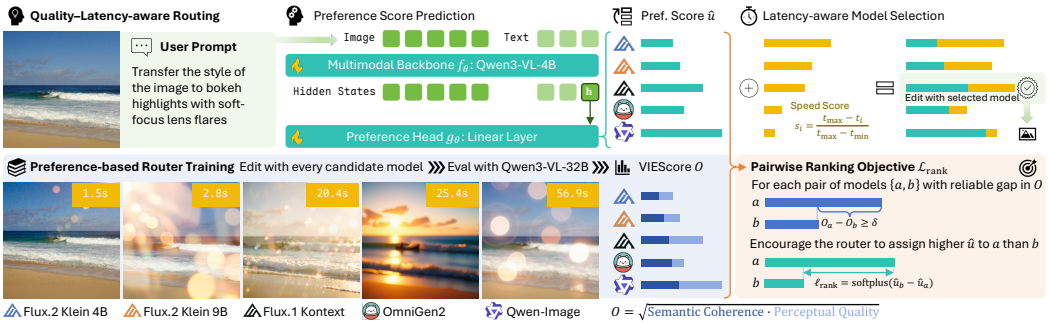

Figure 2: Overview of EditJev. The upper part shows offline candidate generation, automatic evaluation, and router training. The lower part shows quality–latency-aware inference, which combines preference scores with model-level speed scores.

router training and quality–latency-aware inference. Given a source image and an editing instruction, EditJev uses a multimodal large language model (MLLM) backbone to encode the request and the candidate model set, and attaches a lightweight preference head to predict a preference score for each candidate. These preference scores are combined with model-level latency statistics to select one model for execution, enabling quality–latency-aware routing without invoking all candidate models.

Preference Score Prediction. Formally, given a source image x, an editing instruction c, and an ordered candidate model list $\mathcal{M} = (M_1, \ldots, M_K)$, EditJev uses a fixed multimodal routing template to construct the prompt $R(x, c, \mathcal{M})$, which describes the editing request and lists the candidate models. The MLLM backbone $F_{\boldsymbol{\theta}}$ encodes the prompt into hidden states $\mathbf{H} \in \mathbb{R}^{L \times d}$, from which we take the hidden state of the last non-padding token as the request representation:

$$\mathbf{H} = F_{\boldsymbol{\theta}}\big(R(x, c, \mathcal{M})\big), \qquad \mathbf{h} = \mathbf{H}_j \in \mathbb{R}^d, \tag{2}$$

where j denotes the position of the last non-padding token, L denotes the sequence length, and d is the hidden dimension.

A preference head $G_{\boldsymbol{\phi}}$ maps $\mathbf{h}$ to a vector of preference scores over the candidate models:

$$\hat{\mathbf{u}} = G_{\boldsymbol{\phi}}(\mathbf{h}) = (\hat{u}_1, \ldots, \hat{u}_K), \tag{3}$$

where $\hat{u}_i$ denotes the predicted preference score of candidate model M_i. The resulting fixed-dimensional score vector serves as the System One decision output and can be consumed directly by the subsequent model-selection rule without autoregressive decoding or response parsing.

Latency-Aware Model Selection. Although the predicted preference scores characterize the relative preferences among candidate models for the current editing request, they do not account for differences in generation latency across models. To incorporate generation efficiency into the routing decision, we use the mean end-to-end generation latency t_i of each candidate model M_i and convert it into a normalized speed score:

$$s_i = \frac{t_{\max} - t_i}{t_{\max} - t_{\min}}, \tag{4}$$

where $t_{\max}$ and $t_{\min}$ denote the maximum and minimum mean generation latencies among the candidate models, respectively. Consequently, faster models receive larger speed scores, with $s_i \in [0, 1]$, where larger values indicate lower generation latency.

Finally, EditJev combines the preference score with the speed score:

$$\hat{i} = \arg\max_{i \in [K]} \left(\hat{u}_i + \beta s_i\right), \qquad \hat{M} = M_{\hat{i}}, \tag{5}$$

where β controls the quality–latency trade-off and allows training-free adjustment. Only the selected model $\hat{M}$ is executed to produce the final edited image.

3.3 PREFERENCE-BASED ROUTER TRAINING

Overview. To train the editing router, we construct pairwise preference supervision from candidate model outputs to capture the relative suitability of different editors. For each training request, all candidate editing models are executed offline and evaluated by an MLLM-based evaluator along complementary quality dimensions. We then convert the resulting quality scores into preference pairs and optimize the router with a pairwise ranking objective.

Pairwise Preference Construction. For each training request (x, c), we execute every candidate model $M_i \in \mathcal{M}$ to obtain its edited image $y_i = E_i(x, c)$, where E_i denotes the editing function implemented by M_i. Following VIEScore (Ku et al., 2024), we use an MLLM evaluator to assess editing success, preservation of content that should remain unchanged, image naturalness, and freedom from visual artifacts, each on a scale from 0 to 10. The semantic-consistency score SC_i is the minimum of the editing-success and preservation scores, and the perceptual-quality score PQ_i is the minimum of the naturalness and artifact-free quality scores. For this request, we define the aggregate quality score of candidate model M_i as their geometric mean:

$$O_i = \sqrt{SC_i \cdot PQ_i}. \tag{6}$$

Given the aggregate quality scores $\mathbf{O} = (O_1, \ldots, O_K)$, we retain ordered model pairs whose quality-score difference is at least δ:

$$\mathcal{P}(x, c) = \{(a, b) \in [K]^2 : O_a > O_b,\ O_a - O_b \geq \delta\}, \tag{7}$$

For each pair $(a, b) \in \mathcal{P}(x, c)$, candidate model M_a is preferred to candidate model M_b. The score-gap threshold δ excludes near-tied pairs whose relative ordering may be unreliable.

Pairwise Ranking Objective. Given the router prediction $\hat{\mathbf{u}} = G_{\boldsymbol{\phi}}(\mathbf{h}) = (\hat{u}_1, \ldots, \hat{u}_K)$ for request (x, c), we define the pairwise ranking loss as

$$\ell_{\text{rank}}(x, c) = \frac{\sum_{(a,b)\in\mathcal{P}(x,c)} \text{softplus}(\hat{u}_b - \hat{u}_a)}{\max(1, |\mathcal{P}(x, c)|)}, \tag{8}$$

where the sum is defined as zero when $\mathcal{P}(x, c) = \varnothing$. Given a training minibatch $\mathcal{B}$, the overall objective is

$$\mathcal{L}_{\text{rank}} = \frac{1}{|\mathcal{B}|} \sum_{(x,c)\in\mathcal{B}} \ell_{\text{rank}}(x, c). \tag{9}$$

Compared with MCQ-style supervision, our objective avoids compressing the supervision signal into a single best-model label and can exploit multiple reliable pairwise comparisons from one request. It is also more robust than absolute score prediction, as it only relies on reliable relative preferences rather than requiring the router to match the calibrated scale of evaluator scores. Further discussion is provided in Section 4.3.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Implementation Details. We use Qwen3-VL-4B-Instruct (Bai et al., 2025) as the multimodal router backbone, freezing its pretrained parameters and optimizing LoRA adapters and a linear preference head. The candidate pool consists of FLUX.2 Klein 4B, FLUX.2 Klein 9B, FLUX.1-Kontext-dev, OmniGen2, and Qwen-Image-Edit. β is set as 0.7 and 0.02 on ImgEdit and GEdit-En. All latency measurements are conducted on an NVIDIA RTX PRO 6000 GPU using each model's official inference configuration. We measure the generation time of each model over 20 different test requests and report the average. We set the pairwise score-gap threshold to $\delta = 0.5$. The routing latency of each routing method is measured separately as the additional overhead before executing the selected editor. For closed-source routing models, including API-based MLLM routers and Jev, the latency is measured as the API response time and averaged over 5 requests. For EditJev, the latency is measured using local inference and averaged over 5 requests.

Table 1: Quantitative comparison on ImgEdit. Methods are grouped into fixed models, MLLM-based routing methods, and System One routing methods. The best and second-best results for each editing category, overall quality, and latency are marked in bold and underline, respectively. Latency denotes the additional routing overhead before executing the selected editor.

Method	Add	Adjust	Extract	Replace	Remove	Background	Style	Compose	Action	Overall↑	Latency(s)↓
Fixed Models											
FLUX.2-Klein-4B	4.470	4.660	1.690	4.480	4.300	4.470	4.850	3.510	4.800	4.073	–
FLUX.2-Klein-9B	4.580	4.730	1.950	4.680	4.710	**4.680**	**4.910**	3.590	**4.880**	4.252	–
FLUX.1-Kontext-dev	4.050	3.930	1.950	4.440	3.420	3.940	4.400	2.460	4.240	3.670	–
OmniGen2	3.730	3.640	1.640	4.040	3.110	3.680	4.580	2.670	4.420	3.461	–
Qwen-Image-Edit	**4.660**	4.650	**4.120**	4.750	4.630	4.410	4.770	**3.770**	4.750	4.540	–
MLLM Routing											
GPT-5.5 Routing	4.255	4.496	3.994	4.735	4.577	4.304	4.674	3.072	4.509	4.380	11.85
Gemini-3.7-Flash Routing	4.048	4.287	3.906	4.441	3.740	3.964	4.507	3.203	4.639	4.123	17.68
GLM-5.3-Flash Routing	4.132	4.060	3.242	4.634	4.154	4.297	4.753	2.898	4.602	4.137	4.39
Kimi-K3 Routing	4.385	4.578	2.949	4.554	3.980	4.130	4.750	2.956	4.509	4.134	14.50
System One Routing											
Jev Routing	4.349	4.730	2.262	4.543	4.457	4.141	4.830	3.405	**4.880**	4.138	0.87
EditJev	4.543	**4.762**	4.088	**4.772**	**4.716**	4.662	4.890	3.651	4.741	**4.592**	**0.094**

For ablations of the training objective, visual input, and backbone scale, we fix $\beta = 0$ to evaluate their effects on routing quality. We separately vary β to study the inference-time quality–latency trade-off. More details are provided in the appendix.

Compared Methods. We compare EditJev with fixed-model policies, general-purpose MLLM routing baselines, and System-One routing baselines. Fixed-model policies always invoke the same candidate model; we report each candidate separately. For general-purpose MLLM routing, we evaluate GPT-5.5 (OpenAI, 2026), Gemini-3.7-Flash (Google, 2026), GLM-5.3-Flash (Z.AI, 2026), and Kimi-K3 (Moonshot AI, 2026) on the same 737 ImgEdit and 606 GEdit-En requests. Each MLLM receives the source image, editing instruction, candidate model names, and their measured mean generation times through the same prompt template. It selects the editor expected to provide the highest quality, preferring faster editors when the expected quality is similar. Candidate outputs and reference scores are not included in the prompt. Jev is included as a System-One routing baseline that directly predicts preferences over candidate models without autoregressive model selection.

Benchmarks. We evaluate EditJev on the ImgEdit (Ye et al., 2026) and GEdit-En (Liu et al., 2025) image-editing benchmarks, using GPT-5.1 as the primary evaluator and following the official evaluation protocols. We report the benchmark-defined metrics for each dataset.

4.2 Main Results

Quantitative Comparison. Table 1 and 2 compare EditJev with fixed-model policies and general-purpose MLLM routing baselines. On ImgEdit, EditJev achieves an overall score of 4.592, surpassing the strongest fixed-model baseline, Qwen-Image-Edit, which obtains a score of 4.540. On GEdit-En, EditJev achieves a G_O score of 8.255, compared with 8.194 from Qwen-Image-Edit. These results demonstrate that sample-level routing can effectively exploit the complementary capabilities of different editing models rather than relying on a single globally preferred model. Compared with GPT-5.5 Routing, a general-purpose strong MLLM baseline, EditJev achieves higher aggregate scores on both benchmarks, improving from 4.380 to 4.592 on

Table 2: Quantitative evaluation results on GEdit-En.

Method	G_SC	G_PQ	G_O↑
Fixed Models			
FLUX.2-Klein-4B	7.731	8.397	7.667
FLUX.2-Klein-9B	8.081	**8.416**	7.965
FLUX.1 Kontext	6.404	7.579	6.241
OmniGen2	6.391	7.762	6.297
Qwen-Image-Edit	8.441	8.326	8.194
MLLM Routing			
GPT-5.5 Routing	8.238	8.110	7.920
Gemini-3.7-Flash Routing	7.244	7.732	6.996
GLM-5.3-Flash Routing	7.652	7.910	7.402
Kimi-K3 Routing	7.760	8.035	7.543
System One Routing			
Jev Routing	7.913	8.281	7.762
EditJev	**8.452**	8.409	**8.255**

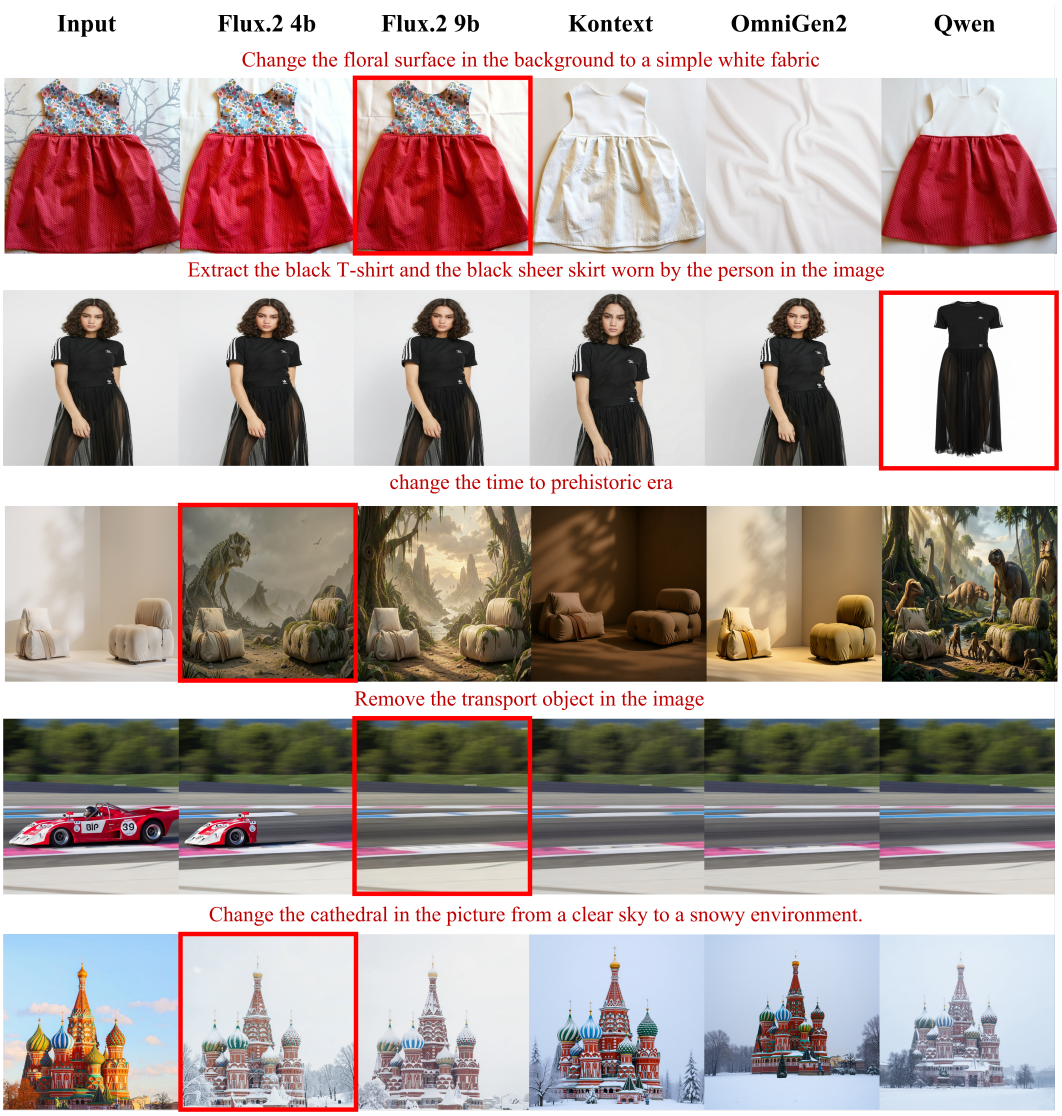

Figure 3: Qualitative routing results on GEdit-En and ImgEdit. A red border marks the model selected by EditJev for each input.

ImgEdit and from 7.920 to 8.255 on GEdit-En. Moreover, following the System One design principle of direct structured decision output, both Jev and EditJev introduce lower routing overhead than general-purpose MLLM routers. While Jev requires 0.87s per request, EditJev further reduces the routing latency to 0.094s per request through a lightweight preference prediction head.

Efficiency Comparison. As shown in Table 3, we compare the end-to-end latency of EditJev with Qwen-Image-Edit and GPT-5.5 Routing, which represent strong fixed-model deployment and general-purpose MLLM-based routing strategies, respectively. The reported latency includes both routing inference and selected-model generation time. Compared with Qwen-Image-Edit, the fixed model with competitive average quality, EditJev reduces the end-to-end latency from 56.9 to 22.3 seconds on ImgEdit and from 56.9 to 38.9 seconds on

Table 3: Latency comparison among Qwen-Image-Edit, GPT-5.5 Routing, and EditJev. Latency includes both the routing inference time and the generation time of the selected editor.

Method	**ImgEdit (s)↓**	**GEdit-En (s)↓**
Qwen-Image-Edit	56.90	56.90
GPT-5.5 Routing	<u>51.15</u>	<u>53.55</u>
EditJev	**22.33**	**38.88**

GEdit-En, achieving $2.55\times$ and $1.46\times$ inference speedups, respectively, while improving the aggregate scores on both benchmarks. This efficiency gain is enabled by EditJev's ability to select faster candidates when their predicted preference scores are comparable, while still selecting slower models when their predicted quality advantages justify the additional latency.

GPT-5.5 Routing achieves end-to-end latencies of 51.15 and 53.55 seconds on ImgEdit and GEdit-En, respectively, which are higher than EditJev's latency. Meanwhile, GPT-5.5 Routing obtains lower aggregate quality scores than both Qwen-Image-Edit and EditJev on both benchmarks (Table 1 and Table 2). Overall, EditJev achieves a better quality–latency trade-off compared with both fixed-model deployment and general-purpose MLLM-based routing strategies.

Qualitative Comparison. Fig. 3 illustrates how EditJev adapts its selection to editing quality and model latency. For the instruction *"Change the floral surface in the background to a simple white fabric"* , only FLUX.2 Klein 4B and FLUX.2 Klein 9B correctly replace the patterned background with simple white fabric, whereas Qwen-Image-Edit, despite being the strongest fixed model on average, incorrectly alters the garment itself. Among the two successful candidates, FLUX.2 Klein 9B better preserves the original garment colors than FLUX.2 Klein 4B, while incurring only a small increase in generation latency, leading EditJev to select it. For the instruction *"Extract the black T-shirt and the black sheer skirt worn by the person in the image,"* only Qwen-Image-Edit successfully extracts the requested garments, and EditJev selects this model despite its higher generation latency. For the instruction *"Remove the transport object in the image"*, all candidates except FLUX.2 Klein 4B successfully remove the racing car. Among these successful candidates, FLUX.2 Klein 9B has the lowest mean generation time and is therefore selected. These examples show that EditJev does not simply choose the strongest or fastest model globally; instead, it selects a model whose output quality is suitable for the specific request while accounting for generation efficiency. This adaptive behavior highlights the advantage of sample-level routing, where different editors can be utilized according to their strengths across diverse editing scenarios.

4.3 Ablation Study

Effect of the Ranking Objective. We compare the pairwise ranking objective with two alternative training formulations. **MSE Regression** replaces the pairwise ranking loss with mean squared error regression on the candidate-quality scores, requiring the router to predict absolute evaluator scores. **MCQ Generation** formulates routing as multiple-choice generation, where the router is trained to output only the index of the highest-scoring candidate model. As shown in Table 4, MSE regression reduces the overall score on ImgEdit from 4.6002 to 4.5545, while MCQ generation further decreases it to 4.5491. These results indicate that pairwise preference learning is more effective for routing than either absolute score regression or single-label generation. The ranking objective directly optimizes the relative ordering among candidate models and can exploit multiple reliable pairwise comparisons from each request. In contrast, MCQ generation compresses the supervision signal into a single best-model label, discarding the preference relationships among the remaining candidates. MSE regression requires the router to accurately predict the evaluator's absolute scores, making the learning problem more challenging.

Table 4: Ablation on the router training objective on the ImgEdit benchmark.

Training Objective	Overall ↑
MSE Regression	4.5545
MCQ Generation	4.5491
Pairwise Ranking (Ours)	**4.6002**

Effect of Multimodal Input. Shown in Table 5, w/o Image Input removes the source image from the router input and performs model selection solely based on the editing instruction. Removing the source image reduces the overall score on ImgEdit from 4.6002 to 4.5717. This result indicates that, even for the same editing instruction, the suitability of different candidate models may vary depending on the source image. Therefore, the router should consider both the editing instruction and the visual content of the source image. Unlike many LLM and text-to-image routing settings that primarily rely on textual inputs, model routing for image editing should jointly consider the editing instruction and the source image.

Table 5: Ablation on the visual input to the router on ImgEdit.

Variant	Overall ↑
w/o Image Input	4.5717
w/ Image Input (Ours)	**4.6002**

Effect of Router Backbone Scale. We further investigate the impact of different router backbone sizes. As shown in Table 6, EditJev achieves comparable routing performance across different Qwen3-VL backbone scales. The results show that increasing the backbone size does not consistently improve routing performance. These results suggest that EditJev is not dependent on a specific backbone scale and can maintain stable routing performance across different multimodal backbones.

Effect of the Speed Weight. We trace the quality–latency trade-off on ImgEdit by varying the inference-time speed weight β from 0.4 to 1.0 in increments of 0.1, while keeping the trained router fixed (Fig. 4). As β increases, estimated average end-to-end latency—computed by averaging the selected models' measured mean generation times and adding router overhead—falls from 26.79 to 17.73 seconds (33.8%), while the overall score changes from 4.5966 to 4.5667. The main setting, $\beta = 0.7$, achieves a score of 4.5916 at 22.33 seconds, only 0.0050 below the score at $\beta = 0.4$. Thus, the speed score allows inference-time adjustment of the quality–latency trade-off without retraining the router.

Table 6: Ablation on the router backbone scale on ImgEdit.

Router Backbone	Overall ↑
Qwen3-VL-2B	4.5898
Qwen3-VL-4B (Ours)	**4.6002**
Qwen3-VL-32B	4.5898

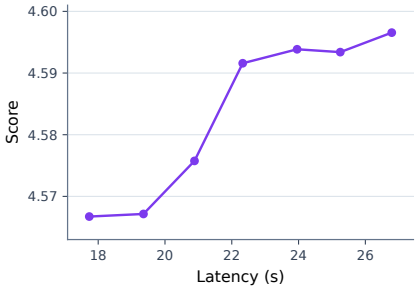

Figure 4: Quality–latency trade-off on ImgEdit under different speed weights.

4.4 User Study

We assess human preference alignment on 20 cases from ImgEdit and GEdit-En. For each case, 20 participants viewed the source image, editing instruction, and outputs of five candidate models alongside the measured mean generation times of the candidate models, then selected a preferred result based on quality and efficiency (400 judgments total). The model selected by EditJev was identical to the participant-selected model in 327 out of 400 individual judgments (81.75%), compared with 138 (34.50%) for GPT-5.5 routing—a 47.25 percentage-point difference (Table 7). Agreement rates are computed over all 400 judgments. We also computed, for each case, the Spearman correlation between candidate vote counts and EditJev's final speed-adjusted routing scores (Eq. 5), including zero-vote candidates and assigning average ranks to ties. The mean correlation across cases was $\rho = \mathbf{0.7478}$, indicating positive alignment between EditJev's ranking and human preferences. We cannot compute a comparable correlation for GPT-5.5 because its recorded responses include only the selected model, not scores or a full ranking.

Table 7: Human preference alignment on 20 cases from ImgEdit and GEdit-En.

Method	Agreement (%) ↑	Spearman ρ ↑
GPT-5.5 Routing	34.50	—
EditJev (Ours)	**81.75**	0.7478

5 Conclusion

To the best of our knowledge, this paper is the first to formulate sample-level image-editing model routing, where a router dynamically selects a suitable editing model for each source image and editing instruction. We introduce EditJev, a System One routing framework that learns preferences over candidate editing models and selects suitable editors for different image–instruction pairs, enabling quality–latency-aware and efficient routing.

Future work will explore more reliable and scalable routing supervision. In particular, we plan to investigate stronger multimodal evaluation frameworks, including agentic multimodal judges that can provide more comprehensive and fine-grained assessments of editing quality, to obtain more reliable preference signals. We also aim to extend routing to larger and dynamically evolving pools of editing models, developing more robust routers that support the addition and removal of editors without extensive retraining. These directions may further improve the scalability and adaptability of image editing model routing systems.

AI USE STATEMENT

In this work, we used generative AI tools to assist with manuscript preparation, including drafting assistance, language refinement, literature discovery and related work exploration, research discussion and exploration, and generating synthetic editing instructions. AI-generated content was carefully reviewed, verified, and revised by the authors. We take full responsibility for the final content of this paper.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 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, 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*, pp. 18392–18402, 2023.

Siyu Cao, Hangting Chen, Peng Chen, Yiji Cheng, Yutao Cui, Xinchi Deng, Ying Dong, Kipper Gong, Tianpeng Gu, Xiusen Gu, et al. HunyuanImage 3.0 technical report. *arXiv preprint arXiv:2509.23951*, 2025.

Lingjiao Chen, Matei Zaharia, and James Zou. FrugalGPT: How to use large language models while reducing cost and improving performance. *Transactions on Machine Learning Research*, 2024.

Tianyu Chen, Yasi Zhang, Zhi Zhang, Peiyu Yu, Shu Wang, Zhendong Wang, Kevin Lin, Xiaofei Wang, Zhengyuan Yang, Linjie Li, et al. Edival-agent: An object-centric framework for automated, fine-grained evaluation of multi-turn editing. *arXiv preprint arXiv:2509.13399*, 2025.

Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Ruehle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *International Conference on Learning Representations*, 2024.

Tao Feng, Yanzhen Shen, and Jiaxuan You. Graphrouter: A graph-based router for llm selections. In *International Conference on Learning Representations*, volume 2025, pp. 26186–26203, 2025.

Tao Feng, Fangxu Yu, Haozhen Zhang, Zhongjie Dai, Liangqi Yuan, Zijie Lei, Weizhi Zhang, Kunlun Zhu, Haodong Yue, Keyang Xuan, et al. Llmrouter: Unified infrastructure for developing, evaluating, and deploying llm routers. *arXiv preprint arXiv:2608.06867*, 2026.

Google. Introducing gemini 3.7 flash. `https://blog.google/innovation-and-ai/models-and-research/gemini-models/introducing-gemini-3-7-flash/`, 2026.

Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. *arXiv preprint arXiv:2208.01626*, 2022.

Qitian Jason Hu, Jacob Bieker, Xiuyu Li, Nan Jiang, Benjamin Keigwin, Gaurav Ranganath, Kurt Keutzer, and Shriyash Kaustubh Upadhyay. Routerbench: A benchmark for multi-llm routing system. *arXiv preprint arXiv:2403.12031*, 2024.

Yi Huang, Jiancheng Huang, Yifan Liu, Mingfu Yan, Jiaxi Lv, Jianzhuang Liu, Wei Xiong, He Zhang, Liangliang Cao, and Shifeng Chen. Diffusion model-based image editing: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 47(6):4409–4437, 2025.

Yuzhou Huang, Liangbin Xie, Xintao Wang, Ziyang Yuan, Xiaodong Cun, Yixiao Ge, Jiantao Zhou, Chao Dong, Rui Huang, Ruimao Zhang, et al. Smartedit: Exploring complex instruction-based image editing with multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8362–8371, 2024.

Mude Hui, Siwei Yang, Bingchen Zhao, Yichun Shi, Heng Wang, Peng Wang, Yuyin Zhou, and Cihang Xie. Hq-edit: A high-quality dataset for instruction-based image editing. *arXiv preprint arXiv:2404.09990*, 2024.

Xu Jiang, Bin Chen, Gehui Li, Yule Duan, Ronggang Wang, and Jian Zhang. Octot2i: A self-evolving agentic text-to-image router. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 31628–31638, 2026a.

Xu Jiang, Gehui Li, Bin Chen, and Jian Zhang. Multi-agent image restoration. *International Journal of Computer Vision*, 134(5):205, 2026b.

Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

Max Ku, Dongfu Jiang, Cong Wei, Xiang Yue, and Wenhu Chen. Viescore: Towards explainable metrics for conditional image synthesis evaluation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12268–12290, 2024.

Guannan Lai, Haoran Hu, and Han-Jia Ye. Routejudge: An open platform for reproducible and preference-aware llm routing. *arXiv preprint arXiv:2606.18774*, 2026.

Zongjian Li, Zheyuan Liu, Qihui Zhang, Bin Lin, Shenghai Yuan, Zhiyuan Yan, Yang Ye, Wangbo Yu, Yuwei Niu, and Li Yuan. Uniworld-v2: Reinforce image editing with diffusion negative-aware finetuning and mllm implicit feedback. *arXiv preprint arXiv:2510.16888*, 2025.

Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, et al. Step1x-edit: A practical framework for general image editing. *arXiv preprint arXiv:2504.17761*, 2025.

Moonshot AI. Kimi k3. `https://www.moonshot.ai/`, 2026.

Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E. Gonzalez, M. Waleed Kadous, and Ion Stoica. RouteLLM: Learning to route LLMs from preference data. In *International Conference on Learning Representations*, 2025.

OpenAI. Introducing gpt-5.5. `https://openai.com/index/introducing-gpt-5-5/`, 2026.

Shelly Sheynin, Adam Polyak, Uriel Singer, Yuval Kirstain, Amit Zohar, Oron Ashual, Devi Parikh, and Yaniv Taigman. Emu Edit: Precise image editing via recognition and generation tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8871–8879, 2024.

Meituan LongCat Team, Hanghang Ma, Haoxian Tan, Jiale Huang, Junqiang Wu, Jun-Yan He, Lishuai Gao, Songlin Xiao, Xiaoming Wei, Xiaoqi Ma, et al. Longcat-image technical report. *arXiv preprint arXiv:2512.07584*, 2025.

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

Yuxiang Tuo, Wangmeng Xiang, Jun-Yan He, Yifeng Geng, and Xuansong Xie. Anytext: Multilingual visual text generation and editing. In *International Conference on Learning Representations*, volume 2024, pp. 56783–56799, 2024.

TypeSafe AI. Introduction. `https://docs.typesafe.ai/introduction`, 2026. Accessed: 2026-09-22.

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*, 2025.

Chenyuan Wu, Jiahao Wang, Pengfei Zheng, Ruiran Yan, Shitao Xiao, Xin Luo, Yueze Wang, Wanli Li, Xiyan Jiang, Yexin Liu, Junjie Zhou, Ziyi Xia, Ze Liu, Chaofan Li, Haoge Deng, Kun Luo, Bo Zhang, Jiajun Zhang, Dong Liu, Defu Lian, Xinlong Wang, Zhongyuan Wang, Tiejun Huang, and Zheng Liu. OmniGen2: Towards instruction-aligned multimodal generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21964–21975, 2026.

Zewei Xin, Qinya Li, Chaoyue Niu, Fan Wu, and Guihai Chen. Adaptive routing of text-to-image generation requests between large cloud model and light-weight edge model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 19482–19491, 2025.

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

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

Qifan Yu, Wei Chow, Zhongqi Yue, Kaihang Pan, Yang Wu, Xiaoyang Wan, Juncheng Li, Siliang Tang, Hanwang Zhang, and Yueting Zhuang. Anyedit: Mastering unified high-quality image editing for any idea. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26125–26135. IEEE, 2025.

Z.AI. Glm-5.3-flash: More intelligence with less compute. `https://autoclaw.z.ai/blog/model/glm-5.3-flash/`, 2026.

Xianghao Zang, Zijian Jiang, Jiarong Cheng, Qianrui Teng, Ying He, Yuxuan Mu, Chao Ban, Huayu Zhang, Lanxiang Zhou, Zerun Feng, et al. Instruction-based image editing: a survey on data, models, evaluation, and applications. *Vicinagearth*, 3(1):3, 2026.

Haozhen Zhang, Tao Feng, and Jiaxuan You. Router-r1: Teaching llms multi-round routing and aggregation via reinforcement learning. *Advances in Neural Information Processing Systems*, 38: 141233–141265, 2026.

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

Haozhe Zhao, Xiaojian Ma, Liang Chen, Shuzheng Si, Rujie Wu, Kaikai An, Peiyu Yu, Minjia Zhang, Qing Li, and Baobao Chang. Ultraedit: Instruction-based fine-grained image editing at scale. *Advances in Neural Information Processing Systems*, 37:3058–3093, 2024.

A Dataset Sources

The router training set contains approximately 53,000 image-instruction pairs. Source images are drawn from ImgEdit (Ye et al., 2026), AnyWord-3M (Tuo et al., 2024), and CelebA-HQ (Karras et al., 2017). Editing instructions are obtained either from the original ImgEdit training set or generated using Qwen3-VL-32B, covering diverse editing scenarios including color alteration, composition, extraction, hybrid editing, material alteration, human-related editing, reference extraction, removal, replacement, style transfer, and tone transfer. For each image-instruction pair, we execute all five candidate editing models to obtain their edited outputs. These outputs are then evaluated using VIEScore (Ku et al., 2024), with Qwen3-VL-32B serving as the multimodal judge. The resulting quality scores are converted into gap-filtered pairwise preferences for router training.

B EditJev Implementation Details

B.1 Training Configuration

EditJev uses a linear preference head with five outputs on top of Qwen3-VL-4B-Instruct. The head takes the hidden state of the last non-padding token and is initialized with zero weights and biases. The pretrained backbone weights are frozen, while the LoRA adapters and preference head are optimized jointly. LoRA is applied to the attention projections q_proj, k_proj, v_proj, and o_proj, as well as the feed-forward projections gate_proj, up_proj, and down_proj. Table 8 lists the training settings.

Table 8: Training configuration for the EditJev model used in the main experiments.

Setting	**Value**
Backbone	Qwen3-VL-4B-Instruct
Preference head	Linear, five outputs
LoRA rank / scaling factor / dropout	64 / 128 / 0.1
Learning rate	5×10^{-6}
Warmup ratio / weight decay	0.05 / 0.01
Training epochs	8
Batch size per GPU / number of GPUs	128 / 2
Gradient accumulation / effective batch size	1 / 256
Backbone computation precision	bfloat16
Random seed	42
Minimum / maximum image pixels	256×28^2 / 512×28^2
Pairwise score-gap threshold δ	0.5

The score-gap threshold δ is used to filter preference pairs rather than weight their loss contributions. For five candidates, at most $\binom{5}{2} = 10$ ordered preference pairs are available for each request. The losses of retained pairs are averaged within each request, followed by averaging across the mini-batch, as in Eq. 9. A request with no retained pairs contributes zero ranking loss. Latency does not enter the training target or ranking loss. Both benchmark evaluations and speed-weight ablations use the saved checkpoint-1040; changing β leaves its parameters fixed.

B.2 Training and Routing Procedure

Algorithm 1 summarizes the complete EditJev procedure. Candidate outputs and evaluator scores are required only when constructing training supervision; at inference time, only the source image, instruction, candidate list, and pre-measured model latencies are used.

B.3 EditJev Input

Source images are converted to RGB before being processed by the native Qwen3-VL processor. As illustrated in Figure 5, the router input consists of the source image, editing instruction, and ordered candidate model list, which form a single user message formatted with the backbone's chat

Algorithm 1 The training and routing procedures of EditJev

Require: Training requests $\mathcal{D}$; candidate editors $\mathcal{M} = (M_1, \ldots, M_K)$; measured latencies $\{t_i\}_{i=1}^{K}$; score-gap threshold δ; speed weight β

Initialize router parameters with frozen backbone, trainable LoRA adapters, and preference head
// 1. Offline Supervision Construction
for each request $(x, c) \in \mathcal{D}$ **do**
  for $i = 1, \ldots, K$ **do**
    Generate candidate output: $y_i \leftarrow E_i(x, c)$
    Evaluate quality score: $O_i \leftarrow \mathrm{Eval}(y_i)$
  end for
  Construct preference pairs: $\mathcal{P}(x, c) \leftarrow \{(a, b) : O_a > O_b,\ O_a - O_b \geq \delta\}$
end for
// 2. Pairwise Router Training
for each minibatch $\mathcal{B} \subset \mathcal{D}$ **do**
  Extract image-instruction representation: $\mathbf{h} \leftarrow F_{\boldsymbol{\theta}}(R(x, c, \mathcal{M}))_j$
  Predict preference scores: $\hat{\mathbf{u}} \leftarrow G_{\boldsymbol{\phi}}(\mathbf{h})$
  Compute ranking loss: $\mathcal{L}_{rank} \leftarrow \frac{\sum_{(a,b)\in\mathcal{P}(x,c)} \mathrm{softplus}(\hat{u}_b - \hat{u}_a)}{\max(1, |\mathcal{P}(x,c)|)}$
  Update LoRA parameters and preference head
end for
// 3. Speed-Aware Inference
Compute normalized speed scores: $s_i \leftarrow \frac{t_{\max} - t_i}{t_{\max} - t_{\min}},\ i = 1, \ldots, K$
For a new request (x, c), predict $\hat{\mathbf{u}}$ with one router forward pass
Select editor: $\hat{i} \leftarrow \arg\max_i(\hat{u}_i + \beta s_i)$
Execute only $M_{\hat{i}}$ to generate the final edited image

Prompt

[IMAGE]

Given the input image and editing instruction, assess which candidate image editing model is most likely to produce the highest-quality edited image.

Editing instruction:
[INSTRUCTION]

Candidate models: flux2_klein_4b, flux2_klein_9b, qwen_image_edit, kontext, omnigen2

Figure 5: Input prompt used by EditJev. The placeholder instruction is replaced by the editing instruction for each input example.

template and its generation-prompt suffix. The preference head directly reads the resulting hidden representation to predict candidate model scores; EditJev does not generate a textual model name.

Line breaks in the displayed template are adjusted for readability. The candidate order defines the order of the five preference-head outputs and their supervision targets. The source image and instruction are the request-specific inputs. Candidate output images and evaluator scores are used only to construct offline supervision and are not provided to EditJev during inference. Model latency is incorporated after preference prediction through Eq. 5.

C Routing Baseline Details

All general-purpose MLLM baselines receive the same editing instructions, candidate model names, candidate order, and measured generation times. For vision-capable MLLM baselines, the source image is provided together with the editing instruction. Jev does not support image input and therefore receives only the textual instruction, candidate names, candidate order, and measured generation times. Its API request is constructed using the same textual selection prompt, with the image compo-

System message

You are a router for instruction-based image editing models.

User message

[IMAGE]

Given an input image and an editing instruction, choose exactly one candidate model to perform the edit.

Candidate models:
1. FLUX.2 Klein 4B, average generation time: 1.5s.
2. FLUX.2 Klein 9B, average generation time: 2.8s.
3. FLUX.1-Kontext-dev, average generation time: 20.4s.
4. OmniGen2, average generation time: 25.4s.
5. Qwen-Image-Edit, average generation time: 56.9s.

All candidates are image editing models. Based on the input image and editing instruction, predict the likely editing result quality of each candidate model. Select the model that is expected to produce the highest-quality edited image for the given input image and instruction. If multiple models are expected to produce similar editing quality, prefer the model with lower average generation time.

Editing instruction:
{INSTRUCTION}

Return only:
{"model": "<model name>"}

Figure 6: Shared selection prompt used for general-purpose MLLM baselines. The placeholder instruction is replaced by the editing instruction for each input example.

nent omitted. All baselines select a single editor without access to candidate outputs or benchmark scores. The shared selection prompt used by general-purpose MLLM baselines is shown in Figure 6.

The prompt design is kept identical across all general-purpose MLLM baselines. All models are prompted to return the selected candidate model in a structured format. The returned model identifier is parsed from the response and mapped to the corresponding candidate model before retrieving its benchmark quality score and generation time. In contrast, Jev directly produces a structured model selection output without autoregressive text generation or response parsing.

Images for vision-capable MLLM baselines are converted to RGB and encoded as JPEG with quality 90 without client-side resizing. GPT-5.5, Gemini-3.7-Flash, GLM-5.3-Flash, and Kimi-K3 use their respective API image processing settings. Gemini, GLM, and Kimi are accessed through the bianxie.ai endpoint, while Jev is accessed through the api.typesafe.ai endpoint.

D Evaluation and Latency Accounting

D.1 Quality Aggregation

For each routing policy, we retrieve the benchmark score of the selected editor for each request and average the scores across all evaluated requests. For GEdit-En, let SC_{ni} and PQ_{ni} denote the semantic consistency and perceptual quality scores of candidate model i for request n. The reported metrics are defined as

$$G_{SC} = \frac{1}{N}\sum_{n=1}^{N} SC_{n\hat{i}_n}, \tag{10}$$

$$G_{PQ} = \frac{1}{N}\sum_{n=1}^{N} PQ_{n\hat{i}_n}, \tag{11}$$

$$G_O = \frac{1}{N}\sum_{n=1}^{N} \sqrt{SC_{n\hat{i}_n} PQ_{n\hat{i}_n}}. \tag{12}$$

Here, G_O is computed by averaging the per-request geometric means rather than by calculating $\sqrt{G_{SC}G_{PQ}}$.

D.2 Candidate Latencies and Speed Scores

Table 9 lists the model-level generation times and normalized speed scores used for latency-aware routing. The same model-level latency is assigned to every request routed to the corresponding candidate editor.

Table 9: Model-level generation times and normalized speed scores used for latency-aware routing.

Candidate editor	Generation time (s)	Speed score
FLUX.2 Klein 4B	1.5	1.0000
FLUX.2 Klein 9B	2.8	0.9765
FLUX.1-Kontext-dev	20.4	0.6588
OmniGen2	25.4	0.5686
Qwen-Image-Edit	56.9	0.0000

For routing decisions $\hat{i}_1, \ldots, \hat{i}_N$, the estimated average selected-editor generation time is

$$\overline{T}_{\text{gen}} = \frac{1}{N}\sum_{n=1}^{N} t_{\hat{i}_n}. \tag{13}$$

The estimated EditJev end-to-end latency is computed as

$$\overline{T}_{\text{EditJev}} = \overline{T}_{\text{gen}} + t_r, \tag{14}$$

where t_r denotes the routing overhead measured for EditJev.

E User Study Details

E.1 Human Preference Metrics

The user study compares router selections with individual participant choices under a joint quality-and-efficiency criterion. Let v_{ni} be the number of participants who choose candidate i on case n, with $\sum_{i=1}^{5} v_{ni} = 20$. For a router that selects $\hat{i}_n$, agreement is computed as

$$\text{Agreement} = \frac{\sum_{n=1}^{20} v_{n\hat{i}_n}}{400}. \tag{15}$$

Since each case receives the same number of responses, this is also the fraction of all 400 judgments that match the router's choices. The recorded counts are 327 for EditJev and 138 for GPT-5.5 Routing, giving 81.75% and 34.50%, respectively. This metric counts individual votes, rather than whether the router matches only the majority winner of a case.

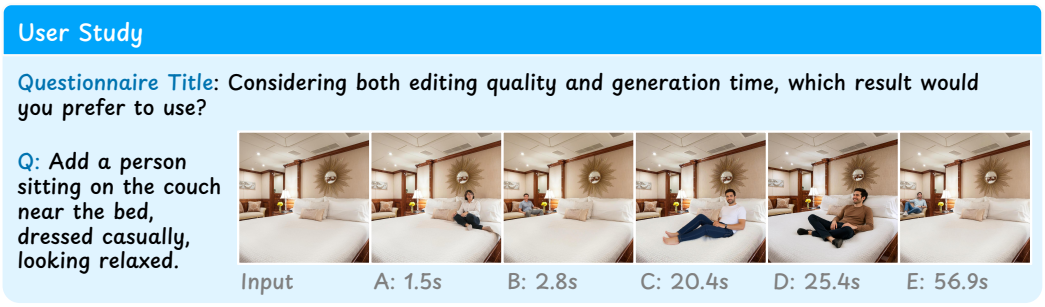

Figure 7: Representative examples from the human preference study. Each example shows the source image, editing instruction, five candidate model outputs, and their corresponding generation times.

For ranking alignment, define $a_{ni} = \hat{u}_{ni} + \beta s_i$ as EditJev's final score. Within each case, we compute Spearman correlation between $(a_{n1}, \ldots, a_{n5})$ and $(v_{n1}, \ldots, v_{n5})$, assigning average ranks to tied values and retaining candidates with zero votes. We then average the 20 within-case correlations, obtaining $\rho = 0.7478$. The retained GPT-5.5 responses specify only one editor and therefore do not provide a complete predicted ranking for the same calculation. Because participants see generation times alongside candidate outputs, these statistics reflect preferences that account for both editing quality and efficiency.

E.2 Human Study Examples

We provide representative examples from the human preference study in Figure 7. Each example presents the source image, editing instruction, five candidate model outputs, and the corresponding generation time of each candidate model. Participants are asked to select the editing result they prefer by considering both editing quality and generation time.