FLOWTOOL: CONTROLLING TOOL PARAMETER IN IMAGE RETOUCHING VIA FLOW MATCHING

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-based image editing (image retouching) is commonly formulated with autoregressive multimodal large language models (MLLMs) that sequentially generate reasoning, tool selections, and parameter values. In this work, we present a novel approach to tool-based image editing by framing the task as a flow matching problem. We introduce FlowTool, a framework that directly models the distribution of high-quality tool parameters conditioned on the input image and user instruction using conditional rectified flow. FlowTool combines a vision-language model backbone for multimodal understanding with a Diffusion Transformer parameter generator that transforms Gaussian noise into an editing plan. We train FlowTool with a two-stage supervised flow-matching curriculum, followed by reward-based post-training. Across MMArt-Bench, FlowTool-Eval, ArtEdit-Bench, and MIT-Adobe5K, FlowTool achieves significantly stronger reference-based performance than specialized MLLM editing agents and proprietary MLLMs, while remaining competitive with proprietary models under reference-free evaluation. Moreover, FlowTool significantly improves inference efficiency, reducing latency by at least $50\times$ while requiring nearly $2\times$ less memory than the compared baselines. These results demonstrate that tool-based image editing can be effectively modeled as conditional generation over structured continuous editing parameters, without autoregressive reasoning.

1 INTRODUCTION

Professional tool-based image editing software provides a rich collection of tools for precise and controllable visual manipulation (Wu et al., 2026). However, effectively using these tools often requires substantial domain expertise. Users must translate high-level editing intents, such as *make the image more cinematic"* or *brighten the subject while preserving the background,"* into concrete editing plans: selecting appropriate tools (e.g., saturation, hue, and opacity) and determining their parameter values. Recent advances in MLLMs (Anthropic, 2026; OpenAI, 2026; Comanici et al., 2025; Wang et al., 2025; Qwen Team, 2026a; Guo et al., 2025) have enabled a promising approach to solving the image retouching (IR) problem. Given an image and an instruction, an MLLM generates reasoning trace, tool selections, and numerical parameter values as sequences of discrete tokens (Lin et al., 2026a; Dutt et al., 2025) in an interactive environment (Yao et al., 2022).

While this formulation is technically convenient, we argue that MLLMs is a fundamental mismatch for the IR problem for 3 reasons: (1) First, IR tools are usually standardized with smooth and continuous linearized scalers for parameters. MLLMs do not yield a high-fidelity, continuous numerical representation sufficient for the sequential decoding of discrete tokens down to numbers (Song & Bahri, 2025). This leads to poor numerical understanding and generation (Lovering et al., 2025; Golkar et al., 2023; Ni, 2026), which is unfit for IR. (2) An MLLM predicts an editing plan token by token, effectively modeling each tool or parameter value conditioned on all preceding actions and parameter predictions Vaswani et al. (2017). This autoregressive formulation exposes the IR task to cascading errors, as mistakes made early in a long prediction trajectory can propagate and compound over subsequent steps (Zhang et al., 2024). We therefore argue that the desired output is better viewed not as a single deterministic sequence of tokens, but as a sample from a conditional joint distribution over coherent editing actions (Chi et al., 2025). (3) Autoregressive agent formulations additionally incur a substantial inference cost. They may generate unnecessarily lengthy reasoning and tool-call trajectories, even though these tokens are not the system's final objective.

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

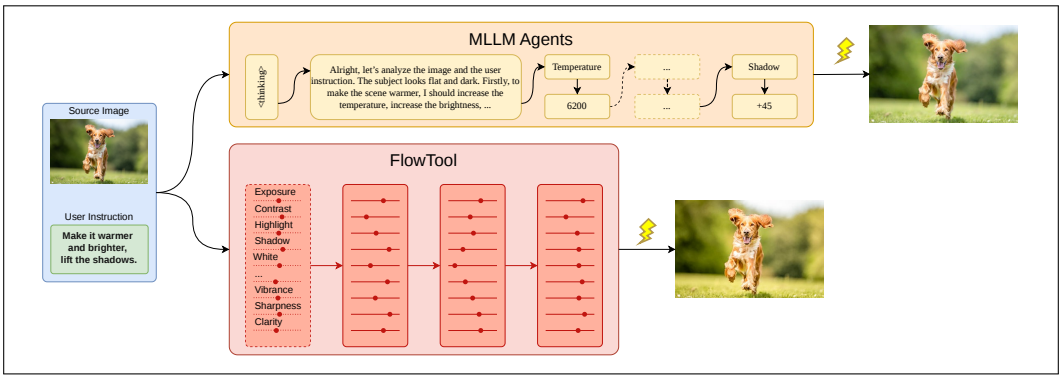

Figure 1: **FlowTool versus autoregressive tool-based image-editing agents.** Existing MLLM agents formulate retouching as language generation, autoregressively producing reasoning, tool selections, and discretized numerical parameters. FlowTool instead treats retouching as conditional generation within a structured, continuous parameter space and directly generates tool parameters via conditional flow matching.

This leads to high latency and memory overhead (Gagrani et al., 2024; Leviathan et al., 2023). This limitation are particularly consequential for interactive editing, where users expect near-real-time feedback, and for deployment on resource-constrained devices.

Motivated by these observations, we formulate tool-based image retouching as a *conditional generative modeling* problem (Liu et al., 2023; Lipman et al., 2023). Given an input image $\mathbf{x}$ and user instruction $\mathbf{c}$, let $\mathbf{a}(\cdot)$ denote an editing plan consisting of the selected tools and their parameters. Rather than representing $\mathbf{a}(\cdot)$ as a language sequence, we directly model the conditional distribution $p(\mathbf{a} \mid \mathbf{x}, \mathbf{c})$ (Chi et al., 2025; Black et al., 2024). This formulation preserves the continuous structure of tool parameters, naturally models the conditional joint distribution over the tool parameter space, and eliminates the need for natural-language reasoning and numerical generation during inference. To this end, we introduce **FlowTool**, a framework that formulates tool-based image editing as conditional flow matching (Tong et al., 2024).

FlowTool combines a VLM backbone for multimodal understanding with a DiT-based tool parameter generator (Peebles & Xie, 2023) for continuous parameter value generation and a tool-presence head for tool selection. We train the model with a two-stage supervised flow-matching curriculum, followed by reward-based post-training (Zheng et al., 2026) that directly optimizes the quality of rendered edits. Across four benchmarks, MMArt-Bench (Lin et al., 2026a), ArtEdit-Bench (Lin et al., 2026b), MIT-Adobe5K (Bychkovsky et al., 2011), and FlowTool-Eval, FlowTool significantly outperforms specialized MLLM agents and proprietary MLLMs on reference-based metrics. Under reference-free evaluation of semantic consistency and perceptual quality, FlowTool also consistently outperforms specialized MLLM agents while remaining competitive with proprietary models. Beyond editing quality, directly generating editing parameters yields substantial computational benefits: FlowTool reduces inference latency by at least $50\times$ and requires nearly $2\times$ less memory than autoregressive baselines. These gains make high-quality tool-based image editing practical in interactive environments and on resource-constrained devices.

We make three key contributions as follows:

- First, we recast tool-based image editing (image retouching) from autoregressive language generation into conditional generative modeling over a structured continuous tool parameter space, better matching the continuous, precision-sensitive, and multimodal nature of professional editing parameters.

- Second, we introduce FlowTool, a conditional flow-matching framework that directly generates tool parameters from multimodal image–instruction representations and further improves rendered outcomes through reward-based post-training.

- Third, we conducted extensive evaluation across four benchmarks, demonstrating that FlowTool achieves strong editing quality while dramatically reducing inference latency and memory consumption compared with autoregressive MLLM agents.

2 Related Work

Tool-based image editing. Early work in tool-based editing predicting editing action to satisfy a general audience without taking into account user intents (Hu et al., 2018; Ke et al., 2022; Ouyang et al., 2023). Recent advancements in MLLMs have led to models that predict editing actions by analyzing user intent (Dutt et al., 2025; Lin et al., 2026a), critiquing images (Wu et al., 2026), and performing iterative editing (Lin et al., 2026b). These methods adopt an autoregressive model to generate both reasoning, tool sequences, and parameter values token-by-token in a discrete space. The development of diffusion models has led to various diffusion-based image editing methods (Duan et al., 2025; Hertz et al., 2022; Liu et al., 2025b; Wu et al., 2023). While these models can handle a broad range of edits, they are highly compute-intensive and often struggle to preserve subject identity Shi et al. (2024). Our work models the editing action space jointly in continuous numerical space through flow matching.

Vision-language-action models. Vision-language-action (VLA) models extend pretrained multimodal representations with action prediction for language-conditioned control (Zitkovich et al., 2023; Kim et al., 2024; Liu et al., 2025a; Wang et al., 2026b; Black et al., 2024; Wang et al., 2026a; Li et al., 2024; Bjorck et al., 2025). In parallel, diffusion-based decision models have established generative action and trajectory modeling as an alternative to autoregressive prediction. Early work models entire trajectories as denoising targets (Janner et al., 2022), with later extensions introducing conditional guidance and inverse-dynamics action recovery (Ajay et al., 2022). More recently, hierarchical diffusion uses language-aligned discrete skill abstractions to condition continuous trajectory generation (Liang et al., 2024), while recent work explores discrete diffusion over tokenized action chunks for non-autoregressive VLA decoding (Liang et al., 2025). FlowTool shares the general principle of conditioning a continuous generator on multimodal representations, but applies it to a fundamentally different problem: generating structured image-editing parameters rather than physical control trajectories.

3 Preliminaries

Flow Matching. Flow matching learns a time-dependent vector field that transports samples between data distribution p_0 and standard Gaussian prior $p_1 = \mathcal{N}(\mathbf{0}, \mathbf{I})$ through the Ordinary Differential Equation (ODE) $\frac{d\mathbf{a}_t}{dt} = \mathbf{v}_\theta(\mathbf{a}_t, t)$ (Lipman et al., 2023). Under the rectified-flow formulation (Liu et al., 2023), data sample $\mathbf{a}_0 \sim p_0$ and Gaussian sample $\mathbf{a}_1 \sim p_1$ are connected by the linear path:

$$\mathbf{a}_t = (1-t)\mathbf{a}_0 + t\mathbf{a}_1, \qquad \mathbf{u}_t = \frac{d\mathbf{a}_t}{dt} = \mathbf{a}_1 - \mathbf{a}_0. \tag{1}$$

The velocity predictor is trained to match this path velocity for t sampled uniformly from $[0, 1]$:

$$\mathcal{L}_{\mathrm{FM}}(\theta) = \mathbb{E}_{\mathbf{a}_0 \sim p_0, \mathbf{a}_1 \sim p_1, t \sim \mathcal{U}[0,1]} \left\| \mathbf{v}_\theta(\mathbf{a}_t, t) - (\mathbf{a}_1 - \mathbf{a}_0) \right\|_2^2. \tag{2}$$

At inference time, sampling starts from $\mathbf{a}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and integrates the learned ODE from $t = 1$ to $t = 0$ to recover a sample from the data distribution. In FlowTool, $\mathbf{a}$ corresponds to the continuous tool-parameter representation, and the velocity field is additionally conditioned on the input image and user instruction, as described in Sec. 4.

Reinforcement Learning for Flow Models. DiffusionNFT (Zheng et al., 2026) performs reward-based post-training through the forward flow-matching process. For condition $\mathbf{y}$, let $\pi_{\mathrm{old}}(\cdot \mid \mathbf{y})$ denote the rollout policy before the current policy update, and let $\hat{\mathbf{a}}_0^{(k)} \sim \pi_{\mathrm{old}}(\cdot \mid \mathbf{y})$ denote the k-th rollout sampled from this policy, with reward $R^{(k)}$. Given K rollouts and the reward-normalization factor $Z_{\mathbf{y}} > 0$, the reward is converted to an optimality weight $r^{(k)} \in [0, 1]$ as:

$$r^{(k)} = \frac{1}{2} + \frac{1}{2}\,\mathrm{clip}\left(\frac{R^{(k)} - \bar{R}}{Z_{\mathbf{y}}}, -1, 1\right), \qquad \bar{R} = \frac{1}{K}\sum_{j=1}^{K} R^{(j)}. \tag{3}$$

For each rollout, we sample $\mathbf{a}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and $t \sim \mathcal{U}[0,1]$, and construct $\hat{\mathbf{a}}_t = (1-t)\hat{\mathbf{a}}_0 + t\mathbf{a}_1$ with target velocity $\hat{\mathbf{u}}_t = \mathbf{a}_1 - \hat{\mathbf{a}}_0$. DiffusionNFT defines

$$\begin{aligned} \mathbf{v}_\theta^+(\hat{\mathbf{a}}_t, t \mid \mathbf{y}) &= (1-\beta)\mathbf{v}_{\text{old}}(\hat{\mathbf{a}}_t, t \mid \mathbf{y}) + \beta \mathbf{v}_\theta(\hat{\mathbf{a}}_t, t \mid \mathbf{y}), \\ \mathbf{v}_\theta^-(\hat{\mathbf{a}}_t, t \mid \mathbf{y}) &= (1+\beta)\mathbf{v}_{\text{old}}(\hat{\mathbf{a}}_t, t \mid \mathbf{y}) - \beta \mathbf{v}_\theta(\hat{\mathbf{a}}_t, t \mid \mathbf{y}), \end{aligned} \tag{4}$$

and optimizes

$$\mathcal{L}_{\text{NFT}} = \mathbb{E}_{\mathbf{y},\, \hat{\mathbf{a}}_0 \sim \pi_{\text{old}}, \mathbf{a}_1 \sim \mathcal{N}(\mathbf{0},\mathbf{I}),\, t \sim \mathcal{U}[0,1]} \left[r \|\mathbf{v}_\theta^+ - \hat{\mathbf{u}}_t\|_2^2 + (1-r)\|\mathbf{v}_\theta^- - \hat{\mathbf{u}}_t\|_2^2 \right]. \tag{5}$$

The positive and negative branches, respectively, encourage high-reward rollouts and discourage low-reward ones. In FlowTool, $\mathbf{y}$ corresponds to the source image and user instruction, and the reward is computed from the image rendered using the predicted editing plan (See Sec. 4.4.2).

4 METHODOLOGY

4.1 PROBLEM FORMULATION

We consider image-editing renderer $\mathcal{R}$ equipped with a collection of editing tools whose behavior is controlled by continuous parameters. Following previous works, we assume that the renderer applies a complex plan with multiple tools using its own static optimal tool order (Lin et al., 2026a; Wu et al., 2026). The continuous parameter space is defined as: $\mathcal{T} = [\ell_1, u_1] \times \cdots \times [\ell_D, u_D]$ where D is the number of continuous tool parameters, and $[\ell_d, u_d]$ specifies the valid *native* range of the d-th parameter. Thus, an element $T_i \in \mathcal{T}$ specifies a valid configuration of the renderer's tools and their associated parameter values.

Given source image $\mathbf{x}$ and natural-language editing instruction $\mathbf{c}$, our goal is to generate an editing plan that can be executed by $\mathcal{R}$ to produce an output image that satisfies the user's intent. The editing plan determines which tools to invoke and specifies their corresponding parameters within $\mathcal{T}$, conditioned on both the visual content of $\mathbf{x}$ and the editing intent expressed by $\mathbf{c}$.

We normalize the i-th parameter value a^i_{raw} to [-1,1] range and represent the editing plan as:

$$\mathbf{a} = [a^1, \ldots, a^D]^\top \in [-1,1]^D, \qquad a^i = 2\frac{a^i_{\text{raw}} - \ell_i}{u_i - \ell_i} - 1. \tag{6}$$

Default values for all the tools are also normalized as $\mathbf{d}^{norm}$. Since an editing plan typically activates only a subset of the available tools, we associate $\mathbf{a}$ with tool-presence mask $\mathbf{m} \in \{0,1\}^D$, where $m^i = 1$ indicates that the i-th tool is active. The corresponding editing plan is:

$$\mathcal{P}(\mathbf{a}, \mathbf{m}) = \text{denorm}(\mathbf{m} \odot \mathbf{a} + (1-\mathbf{m}) \odot \mathbf{d}^{norm}), \tag{7}$$

where the element-wise operator denorm_i maps the normalized a^i back to its native range $[\ell_i, u_i]$.

Let $\mathbf{y} = (\mathbf{x}, \mathbf{c})$ be the multimodal condition. Because multiple parameter configurations may produce valid edits for the same image-instruction pair, we model the conditional distribution $p_0(\mathbf{a}_0 \mid \mathbf{y})$ rather than regressing to a single deterministic solution. For $\mathbf{a}_0 \sim p_0(\cdot \mid \mathbf{y})$ and Gaussian noise $\mathbf{a_1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, we define

$$\mathbf{a}_t = (1-t)\mathbf{a}_0 + t\mathbf{a_1}, \qquad t \in [0,1], \tag{8}$$

with target velocity $\mathbf{a_1} - \mathbf{a}_0$. FlowTool learns the conditional velocity field $\mathbf{v}_\theta(\mathbf{a}_t, t \mid \mathbf{y})$ over the continuous tool parameter space, while a dedicated *tool-presence head* predicts the tool-presence mask $\mathbf{m}$ conditioned on the same multimodal representation.

At inference time, we sample $\mathbf{a}_1 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and integrate the learned ODE from $t = 1$ to $t = 0$ to obtain $\hat{\mathbf{a}}_0$. The tool-presence head predicts $\hat{\mathbf{m}}$, and together $(\hat{\mathbf{a}}_0, \hat{\mathbf{m}})$ defines the editing plan $\mathcal{P}(\hat{\mathbf{a}}_0, \hat{\mathbf{m}})$, and the final image is produced as $\hat{\mathbf{x}} = \mathcal{R}(\mathbf{x}, \mathcal{P}(\hat{\mathbf{a}}_0, \hat{\mathbf{m}}))$. Thus, FlowTool directly generates a structured editing plan in continuous parameter space rather than serializing tool selections, parameter values, and intermediate reasoning into an autoregressive language sequence.

4.2 MODEL ARCHITECTURE

FlowTool consists of three main components: a vision-language model (VLM) backbone for multimodal understanding, a DiT-based tool parameter generator for continuous parameter prediction,

and a tool-presence head for determining which tool (editing parameters) should be activated. Together, these components map an input image-instruction pair to an editing plan. More details of the model architecture and diagram are presented in Fig. 5 of App. A.

Vision-Language Backbone. Given source image $\mathbf{x}$ and editing instruction $\mathbf{c}$, the VLM produces multimodal tokens $\mathbf{H} = \mathrm{VLM}(\mathbf{x}, \mathbf{c})$, which encode both the visual information of the source image and the editing intent. This multimodal representation provides the condition for both parameter generation and tool-presence prediction.

DiT-based Tool Parameter Generator. The parameter generator models the conditional flow over the continuous editing parameters. Given noisy parameter vector $\mathbf{a}_t \in \mathbb{R}^D$, each scalar parameter is projected into the DiT hidden space to form a sequence of parameter representations, $\mathbf{P}_t = \mathrm{Embed}_{\mathrm{param}}(\mathbf{a}_t)$. Multimodal tokens $\mathbf{H}$ and parameter representations $\mathbf{P}_t$ are concatenated and processed by a stack of two-stream DiT blocks, $\left(\mathbf{H}^{\ell+1}, \mathbf{P}_t^{\ell+1}\right) = \mathcal{B}_\ell\left(\mathbf{H}^\ell, \mathbf{P}_t^\ell, t\right)$, where joint attention allows each parameter representation to condition on both the multimodal context and the remaining editing parameters, while the flow timestep t is injected through adaptive normalization. The final parameter representations are projected to the conditional velocity field:

$$\mathbf{v}_\theta(\mathbf{a}_t, t \mid \mathbf{x}, \mathbf{c}) = \mathrm{Head}_{\mathrm{vel}}\left(\mathbf{P}_t^N\right) \in \mathbb{R}^D. \tag{9}$$

Integrating this velocity field from $t = 1$ to $t = 0$ transforms Gaussian noise into the clean tool parameters $\hat{\mathbf{a}}_0$.

Tool-Presence Head. Since a user instruction typically activates only a subset of the available tools, FlowTool additionally predicts a tool-presence mask $\hat{\mathbf{m}} \in \{0, 1\}^D$. The tool-presence head takes multimodal representation $\mathbf{H}$ as input and predicts whether each parameter should be included in the editing plan. Predicted parameter vector $\hat{\mathbf{a}}_0$ and tool-presence mask $\hat{\mathbf{m}}$ jointly define the editing plan $\mathcal{P}(\hat{\mathbf{a}}_0, \hat{\mathbf{m}})$ introduced in Sec. 4.1.

4.3 Data Preparation

We curate the dataset in which each sample is $(\mathbf{x}, \mathbf{x}^*, \mathbf{c}, \mathbf{a}_0, \mathbf{m})$, where $\mathbf{x}$ and $\mathbf{x}^*$ are the source and expert-edited images, respectively, $\mathbf{c}$ is the synthetic user instruction, and $(\mathbf{a}_0, \mathbf{m})$ is the ground-truth editing plan with normalized tool parameter value $\mathbf{a}_0$ and tool-presence mask $\mathbf{m}$.

Expert Editing Data Curation. Using our internal platform, we curate high-quality editing samples consisting of a source image, an edited image, and the tool parameter settings used by experts to produce the edit. We discard incomplete records and map the recorded settings to our fixed tool catalog. The parameter values are normalized to obtain $\mathbf{a}_0$, with unused tools assigned their default parameter values, while the parameters present in the expert edit define tool-presence mask $\mathbf{m}$.

Instruction Synthesis. Our curated dataset does not contain the user instructions. To address this, we use a pretrained VLM to synthesize the instructions. Given source image $\mathbf{x}$, expert-edited image $\mathbf{x}^*$, and recorded parameter values $\mathbf{a_0}$, the VLM infers the intended visual transformation and generates corresponding instruction $\mathbf{c}$. We generate multiple instruction variants for each edit and sample one variant per training example in each epoch. Unlike autoregressive MLLM-based editing agents, FlowTool requires no reasoning traces, reducing the burden of data generation.

4.4 Training

FlowTool is trained in two phases: we first learn the distribution of expert editing plans through a two-stage supervised flow-matching training (SFT phase), and then apply reward-based post-training to directly optimize the rendered editing results (RL phase). Each phase serves a different purpose: the SFT phase establishes reliable tool-parameter generation and tool-presence prediction, while the RL phase allows the model to move beyond exact imitation of the demonstrated expert tool parameters.

4.4.1 Supervised Flow-Matching Training

Training Objective. Given training tuple $(\mathbf{x}, \mathbf{c}, \mathbf{a}_0, \mathbf{m})$, we sample $\mathbf{a_1} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ and construct $\mathbf{a}_t$ following Eq. (8). The DiT-based parameter generator is trained to predict target velocity $\mathbf{u}_t =$

$\mathbf{a_1} - \mathbf{a}_0$. Since only tools present in the expert editing plan should contribute to parameter regression, we compute the flow-matching loss over the active tools indicated by mask $\mathbf{m}$:

$$\mathcal{L}_{\mathrm{FM}} = \mathbb{E} \frac{\left\| \sqrt{\mathbf{m}} \odot (\mathbf{v}_\theta(\mathbf{a}_t, t \mid y) - \mathbf{u}_t) \right\|_2^2}{\max(1, \mathbf{1}^\top \mathbf{m})}. \tag{10}$$

In parallel, the tool-presence head predicts whether each parameter is active in the editing plan. We train it using a binary cross-entropy objective:

$$\mathcal{L}_{\mathrm{pres}} = -\frac{1}{D} \sum_{i=1}^{D} \left[m^i \log \hat{m}^i + (1 - m^i) \log(1 - \hat{m}^i) \right], \tag{11}$$

where $\hat{m}^i$ is the predicted presence probability for the i-th tool. The complete supervised objective is the sum of regular flow matching loss and the weighted cross-entropy loss for tool masking prediction parameterized by λ_{pres}:

$$\mathcal{L}_{\mathrm{SFT}} = \mathcal{L}_{\mathrm{FM}} + \lambda_{\mathrm{pres}} \mathcal{L}_{\mathrm{pres}}. \tag{12}$$

Two-stage SFT Curriculum. The VLM backbone is already pretrained, whereas the DiT parameter generator and tool-presence head are randomly initialized. Jointly optimizing all components from the beginning can cause unstable updates from the randomly initialized modules to propagate into the VLM, corrupting its pretrained multimodal representations before the new components have learned meaningful task structure. We therefore adopt a two-stage curriculum. We first freeze the VLM and train only the parameter generator and tool-presence head, allowing the newly introduced modules to first acquire the tool-parameter distribution and tool-activation patterns under stable pretrained conditioning. Then, we enable LoRA adapters in the VLM and jointly optimize all trainable components. This second stage allows the VLM representations and the prediction modules to adapt to one another, refining the multimodal features toward fine-grained editing decisions while preserving the knowledge acquired during the first stage.

4.4.2 Reward-Based Post-Training

Intuition. The SFT phase encourages FlowTool to reproduce the demonstrated expert editing plans. However, the expert parameter setting represents only one possible solution, and different parameter configurations can produce edits that satisfy the same user request. We therefore further optimize the SFT model using DiffusionNFT (Zheng et al., 2026), as introduced in Sec. 3. For each image–instruction pair, the model samples a group of editing plans, which are executed by the renderer to obtain edited images $\{\hat{\mathbf{x}}^{(k)}\}_{k=1}^{G}$. The resulting images are scored and used to construct the group-relative rewards for training.

Reward Design. Our total reward is a weighted sum of reference-based reward λ_{ref} and reference-free reward λ_{vlm}:

$$R^{(k)} = \lambda_{\mathrm{ref}} R^{(k)}_{\mathrm{ref}} + \lambda_{\mathrm{vlm}} R^{(k)}_{\mathrm{vlm}}. \tag{13}$$

The reference-based reward measures similarity to expert-edited target $\mathbf{x}^*$ using the negative pixel-wise L_1 distance. The negative sign converts the distance into a reward, such that outputs closer to the expert rendition receive higher scores as follows:

$$R^{(k)}_{\mathrm{ref}} = -\|\hat{\mathbf{x}}^{(k)} - \mathbf{x}^*\|_1. \tag{14}$$

Because matching a single expert rendition does not fully capture whether an edit satisfies the user's request, we additionally employ a VLM-as-a-judge reward. Given original image $\mathbf{x}$, rendered output $\hat{\mathbf{x}}^{(k)}$, and user instruction $\mathbf{c}$, the VLM judge evaluates how well the transformation from $\mathbf{x}$ to $\hat{\mathbf{x}}^{(k)}$ fulfills the editing instruction:

$$R^{(k)}_{\mathrm{vlm}} = \mathrm{VLMJudge}(\mathbf{x}, \hat{\mathbf{x}}^{(k)}, \mathbf{c}). \tag{15}$$

Together, the two rewards encourage FlowTool to remain consistent with expert editing outcomes while directly optimizing whether the rendered result fulfills the user's instructions.

Table 1: **Performance comparison on 4 datasets**. ↓ indicates lower is better and ↑ means higher is better. The best and second-best for each column are shown in **bold** and underlined, respectively. The Δ rows report the relative percentage improvement (in green) and regression (in red) of the best-performing FlowTool variant (SFT or RL) against the strongest general/specialized MLLMs.

Method	MMArt-Bench					FlowTool-Eval					ArtEdit-Bench					MIT-Adobe5K		
	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑	PSNR ↑	SSIM ↑	LPIPS ↓
GPT-5.6 Sol	10.77	22.68	**8.34**	9.38	8.80	8.81	16.44	8.27	9.55	**8.84**	8.73	16.60	8.37	9.53	8.88	17.25	**0.52**	0.38
Gemini 3.1 Pro	12.16	27.65	8.14	9.22	8.63	10.23	21.10	**8.30**	9.41	8.81	9.96	21.82	8.31	9.39	8.78	16.88	0.45	0.42
Claude Sonnet 5	10.92	22.57	8.20	9.35	8.72	9.47	18.47	7.68	9.56	8.50	8.77	16.53	8.34	9.50	8.86	17.58	0.47	0.41
Claude Opus 4.8	12.13	27.48	8.26	9.28	8.73	9.38	17.95	8.19	9.48	8.78	8.97	16.99	**8.44**	9.47	8.90	17.56	0.46	0.37
JarvisArt	11.64	28.69	7.31	9.25	8.13	11.06	25.12	7.31	9.44	8.19	9.56	22.72	7.87	9.46	8.56	17.39	0.47	0.40
RetouchIQ	17.94	60.80	6.42	9.01	7.45	16.83	58.64	7.01	9.16	7.92	15.24	48.83	7.41	9.07	8.12	14.16	0.39	0.52
FlowTool-SFT	**9.37**	**17.78**	7.96	**9.59**	8.69	8.24	14.65	7.53	**9.65**	8.41	7.90	13.92	8.13	**9.78**	8.87	17.83	**0.52**	**0.36**
FlowTool-RL	9.42	17.98	8.16	**9.59**	**8.81**	**7.87**	**13.21**	8.04	9.64	8.76	**7.65**	**12.95**	8.34	9.75	**8.99**	**17.90**	0.51	**0.36**
Δ *vs. General*	+13.0%	+21.2%	-2.2%	+2.2%	+0.1%	+10.7%	+19.6%	-3.1%	+0.9%	-0.9%	+12.4%	+21.7%	-1.2%	+2.6%	+1.0%	+1.8%	0.0%	+2.7%
Δ *vs. Specialized*	+19.5%	+38.0%	+11.6%	+3.7%	+8.4%	+28.8%	+47.4%	+10.0%	+2.2%	+7.0%	+20.0%	+43.0%	+6.0%	+3.4%	+5.0%	+2.9%	+10.6%	+10.0%

5 EXPERIMENTS

5.1 EXPERIMENTAL SETTINGS

Models and Training Data. We use Qwen3.5-4B (Qwen Team, 2026a) as the VLM backbone of the FlowTool model. During reward-based post-training, we utilize Qwen3.8-27B (Qwen Team, 2026b) as the VLM judge. Our training corpus is constructed from an internally curated collection of expert editing records. Following Sec. 4.3, we synthesize user instructions using Gemma4-31B (Team et al., 2026), resulting in 496,693 training examples. Finally, FlowTool uses three ODE steps at inference time by default.

Benchmarks. We evaluate FlowTool on three established tool-based image-editing benchmarks, MMArt-Bench (Lin et al., 2026a), ArtEdit-Bench (Lin et al., 2026b), and MIT-Adobe5K (Bychkovsky et al., 2011). We additionally evaluate on **FlowTool-Eval**, which contains 300 held-out examples from our internally curated dataset, with no overlap with the training data.

Evaluation Metrics. Following standard practice in prior work (Lin et al., 2026a; Wu et al., 2026), we report both reference-based and reference-free metrics. For reference-based evaluation, we measure agreement with the expert-edited target using PSNR, SSIM, and LPIPS (Zhang et al., 2018) on MIT-Adobe5K, and L_1 and L_2 on other benchmarks. For reference-free evaluation, we report Semantic Consistency (SC), Perceptual Quality (PQ), and Overall Score (O) (Ku et al., 2024) using GPT-5 (OpenAI, 2025) as the judge.

Baselines. We compare FlowTool with two groups of baselines. Firstly, we evaluate proprietary MLLMs, including GPT-5.6 Sol, Gemini 3.1 Pro, Claude Sonnet 5, and Claude Opus 4.8, prompting them to generate editing plans without task-specific training. Secondly, we compare against specialized MLLM agents: JarvisArt (Lin et al., 2026a) and RetouchIQ (Wu et al., 2026).

5.2 RESULTS & ANALYSIS

Quantitative Comparison. As shown in Table 1, FlowTool consistently outperforms specialized MLLM editing agents across all benchmarks, with the largest gains on reference-based metrics: compared with the strongest specialized baseline, it reduces L_1 by up to 28.8% and L_2 by up to 47.4%. FlowTool also surpasses the frontier models, reducing L_1 and L_2 by up to 13.0% and 21.7%, respectively, while achieving stronger perceptual quality and remaining competitive in semantic consistency. Reward-based post-training further improves instruction alignment, with FlowTool-RL increasing SC and the overall score across all three instruction-driven benchmarks. On MIT-Adobe5K, FlowTool outperforms all baselines.

Qualitative Analysis. Fig. 2 reveals clear differences in the editing behavior of the compared methods. Specialized MLLM agents tend to apply extremely aggressive transformations, often producing excessive changes in exposure, color, or contrast that deviate from the intended edit. This behavior is particularly evident for JarvisArt in the frog example, where severe overexposure essentially destroys the image content. Similar tendencies can be observed in other examples, where the specialized agents produce overly saturated, excessively dark, or strongly color-shifted outputs. Gemini 3.1

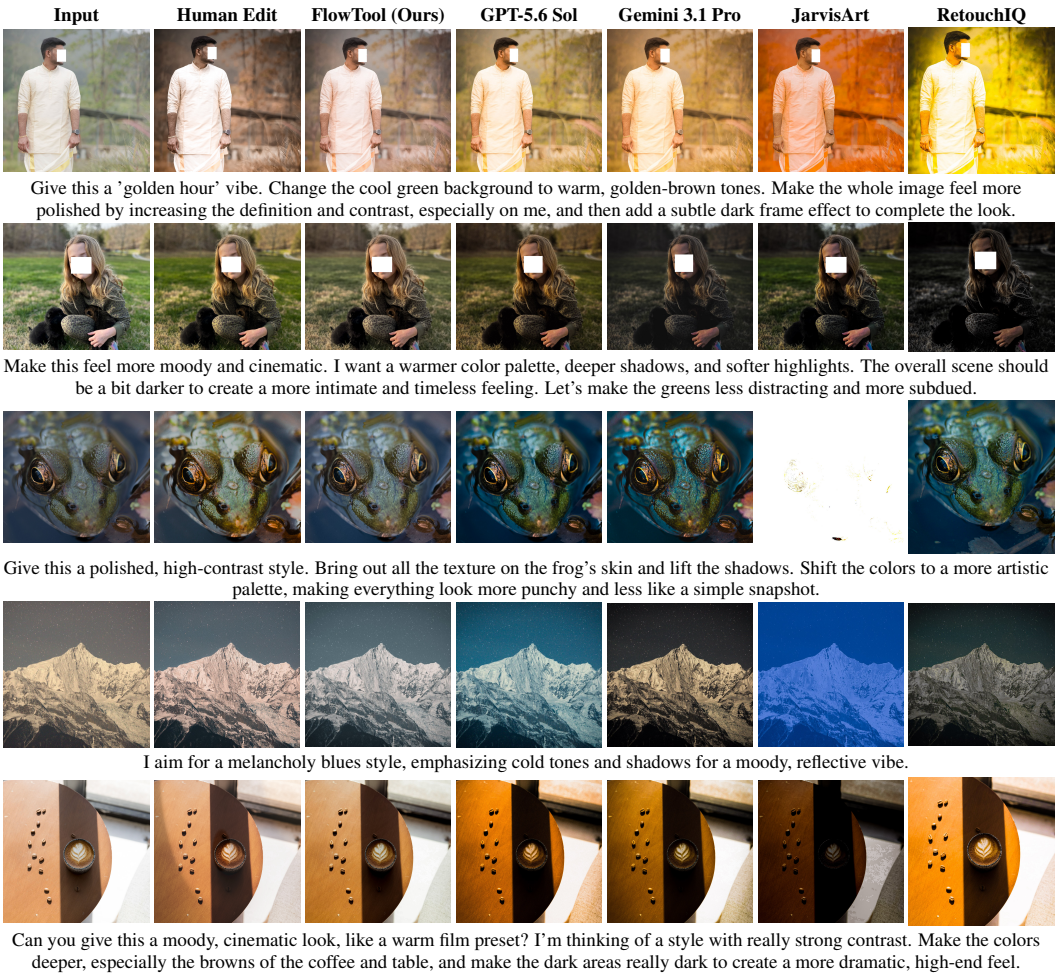

Figure 2: **Qualitative comparison between different methods**. For each example, we show the input image, the user edit, and the outputs produced by FlowTool, proprietary MLLMs, and specialized MLLM agents. The editing instruction for each example is shown below each row.

Pro generally produces more reasonable edits than the specialized agents, but its behavior remains less consistent across instructions; for example, some outputs deviate noticeably from the requested color tone or editing intensity. In contrast, FlowTool produces edits that closely resemble the corresponding human edits while faithfully following the requested changes in tone, color, contrast, and overall appearance. Finally, FlowTool achieves visual quality comparable to GPT-5.6 Sol across these examples, consistent with the quantitative findings.

Efficiency Comparison. We randomly sample 200 examples in total, with 50 examples drawn from each of the four evaluation benchmarks, and perform single-sample inference on a NVIDIA A100 GPU. For locally deployed models, we measure the time required to generate the editing plan and record peak GPU memory usage during inference. Proprietary MLLMs are accessed through their APIs under the same evaluation setup; therefore, we report end-to-end API response time as latency, while their GPU memory usage is unavailable. As shown in Fig. 3, FlowTool is consistently faster than all compared methods, achieving over $50\times$ lower latency than the fastest baselines, while

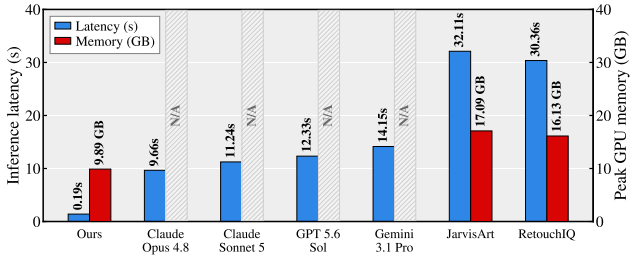

Figure 3: Inference efficiency comparison.

requiring nearly 2× lower peak GPU memory than the specialized MLLM agents. Importantly, these efficiency gains are achieved while maintaining strong editing performance across our benchmarks.

5.3 Ablation Study

To analyze the effect of FlowTool's individual components and design choices, we conduct all ablation experiments on MMArt-Bench.

Effect of ODE Integration Steps. Fig. 4 shows that increasing the number of steps from one to two substantially improves performance across reference-based and reference-free metrics. Performance improves at three steps, providing the best balance across L_1, L_2, SC, and PQ. Increasing the number of integration steps beyond this point brings little additional benefit. These results indicate that FlowTool can accurately generate editing parameters with only a few ODE steps, whereas a single step (i.e., a simple regression model) does not yield a sufficiently good image-editing plan.

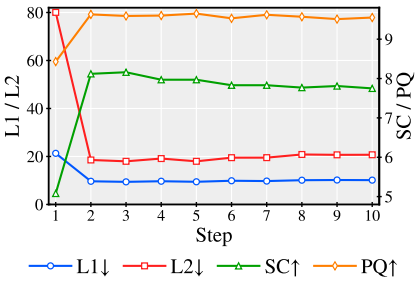

Figure 4: Effect of #ODE steps.

Tool-Presence Modeling. Table 2 shows that removing the tool-presence head degrades performance, for both L_1 and L_2, while causing a pronounced drop in SC and the overall score. This demonstrates that explicitly modeling which parameters should be activated is important for producing accurate editing plans.

Table 2: Ablation of the tool-presence head.

Setting	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑
w/o Head	9.56	18.37	7.04	**9.61**	8.07
w/ Head	**9.37**	**17.78**	**7.96**	9.59	**8.69**

Two-Stage SFT Curriculum. As shown in Table 3, jointly training all components from the beginning yields substantially worse performance, particularly for SC and the overall score. We attribute this degradation to the optimization imbalance between the pretrained VLM backbone and the randomly initialized DiT parameter generator and tool-presence head, whose unstable early updates can interfere with the pretrained multimodal representations. Training only the randomly initialized components, while keeping the VLM frozen, already results in stronger and more stable performance. Our full two-stage curriculum performs best overall: it first establishes reliable parameter generation and tool-presence prediction with the VLM frozen, and only then adapts the VLM through joint training, enabling its pretrained representations to specialize toward fine-grained editing decisions.

Table 3: Ablation of the two-stage SFT curriculum.

SFT Schedule	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑
Stage 1 only	9.63	18.92	7.73	9.55	8.50
Stage 2 only	9.90	20.10	5.99	**9.69**	7.42
Stage 1 → Stage 2	**9.37**	**17.78**	**7.96**	9.59	**8.69**

Scaling the Backbone Table 4 shows that scaling the VLM backbone from 0.8B to 4B generally improves performance, with Qwen3.5-4B achieving the strongest overall results. Further scaling the backbone to 9B provides no additional improvement and slightly degrades most metrics. We therefore adopt Qwen3.5-4B as our default backbone.

Table 4: Ablation of VLM backbone scale.

VLM Backbone	L1 ↓	L2 ↓	SC ↑	PQ ↑	O ↑
Qwen3.5-0.8B	9.58	18.65	7.27	9.61	8.25
Qwen3.5-4B	**9.37**	**17.78**	**7.96**	9.59	**8.69**
Qwen3.5-9B	9.44	18.20	7.54	**9.63**	8.44

6 Conclusion

We introduced **FlowTool**, a novel approach to tool-based image editing that replaces autoregressive reasoning with direct structured tool-parameter generation through conditional flow matching. By combining a VLM backbone for multimodal understanding with a DiT-based tool parameter generator, FlowTool directly produces editing parameters conditioned on the source image and user instruction. Extensive experiments across multiple tool-based image editing benchmarks demonstrate that FlowTool achieves strong editing performance, outperforming specialized MLLM agents while remaining competitive with frontier proprietary MLLMs. At the same time, FlowTool provides substantially more efficient inference in terms of both latency and memory usage.

AI USE STATEMENT

We used large language models (LLMs) in several supporting tasks of this work. We used LLMs as assistants during codebase development for coding and debugging. All AI-assisted code was reviewed, tested, and validated by the authors. We also used LLMs to assist with paraphrasing, grammatical correction, and improving the clarity and readability of our writing.

The research ideas, methodology, experimental design, analysis, scientific claims, and the paper's overall structure and content were developed and determined by the authors. LLMs were not used to autonomously formulate scientific contributions or draw conclusions from experimental results. All AI-assisted outputs were reviewed and, where necessary, revised by the authors. We take full responsibility for the final content of the paper and all associated artifacts.

ETHICS STATEMENT

We do not involve any human subjects or data annotation throughout the project. Datasets were sourced from licensed open-source datasets and licensed private datasets.

REPRODUCIBILITY STATEMENT

Secs. 3 and 4 describe the method including conceptual design, mathematical formulation. Figs. 1 and 5 illustrate the model architecture. Sec. 4.3 describes the data preparation process. Computational cost accounting is presented in Sec. A. Accessing to datasets, model checkpoints, and code are subjected to internal approval.

REFERENCES

Anurag Ajay, Yilun Du, Abhi Gupta, Joshua Tenenbaum, Tommi Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *arXiv preprint arXiv:2211.15657*, 2022.

Anthropic. Introducing claude fable 5.1 and claude mythos 5.1. `https://www.anthropic.com/claude-fable-and-mythos-5-1`, 2026.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, 44(10-11):1684–1704, 2025.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Zheng-Peng Duan, Jiawei Zhang, Zheng Lin, Xin Jin, XunDong Wang, Dongqing Zou, Chun-Le Guo, and Chongyi Li. Diffretouch: Using diffusion to retouch on the shoulder of experts. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 2825–2833, 2025.

Niladri Shekhar Dutt, Duygu Ceylan, and Niloy J Mitra. Monetgpt: Solving puzzles enhances mllms' image retouching skills. *ACM Transactions on Graphics (TOG)*, 44(4):1–12, 2025.

Mukul Gagrani, Raghavv Goel, Wonseok Jeon, Junyoung Park, Mingu Lee, and Christopher Lott. On speculative decoding for multimodal large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 8285–8289, June 2024.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, et al. xval: a continuous numerical tokenization for scientific language models. *arXiv preprint arXiv:2310.02989*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

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.

Yuanming Hu, Hao He, Chenxi Xu, Baoyuan Wang, and Stephen Lin. Exposure: A white-box photo post-processing framework. *ACM Transactions on Graphics (TOG)*, 37(2):1–17, 2018.

Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022.

Zhanghan Ke, Chunyi Sun, Lei Zhu, Ke Xu, and Rynson WH Lau. Harmonizer: Learning to perform white-box image and video harmonization. In *European conference on computer vision*, pp. 690–706. Springer, 2022.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

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.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International conference on machine learning*, pp. 19274–19286. PMLR, 2023.

Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, et al. Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation. *arXiv preprint arXiv:2411.19650*, 2024.

Zhixuan Liang, Yao Mu, Hengbo Ma, Masayoshi Tomizuka, Mingyu Ding, and Ping Luo. Skilldiffuser: Interpretable hierarchical planning via skill abstractions in diffusion-based task execution. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 16467–16476. IEEE, 2024.

Zhixuan Liang, Yizhuo Li, Tianshuo Yang, Chengyue Wu, Sitong Mao, Liuao Pei, Tian Nian, Shunbo Zhou, Xiaokang Yang, Jiangmiao Pang, et al. Discrete diffusion vla: Bringing discrete diffusion to action decoding in vision-language-action policies. *arXiv preprint arXiv:2508.20072*, 2025.

Yunlong Lin, Zixu Lin, Kunjie Lin, Jinbin Bai, Panwang Pan, Chenxin Li, Haoyu Chen, Zhongdao Wang, Xinghao Ding, Wenbo Li, et al. Jarvisart: Liberating human artistic creativity via an intelligent photo retouching agent. *Advances in Neural Information Processing Systems*, 38: 52088–52130, 2026a.

Yunlong Lin, Linqing Wang, Kunjie Lin, Zixu Lin, Kaixiong Gong, Wenbo Li, Bin Lin, Zhenxi Li, Shiyi Zhang, Yuyang Peng, et al. Jarvisevo: Towards a self-evolving photo editing agent with synergistic editor-evaluator optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 27291–27302, 2026b.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *The Eleventh International Conference on Learning Representations*, 2023.

Jiaming Liu, Hao Chen, Pengju An, Zhuoyang Liu, Renrui Zhang, Chenyang Gu, Xiaoqi Li, Ziyu Guo, Sixiang Chen, Mengzhen Liu, et al. Hybridvla: Collaborative diffusion and autoregression in a unified vision-language-action model. *arXiv preprint arXiv:2503.10631*, 2025a.

Jiaxin Liu, Qichao Ying, Zhenxing Qian, Sheng Li, Runqi Zhang, Jian Liu, and Xinpeng Zhang. Mofrr: Mixture of diffusion models for face retouching restoration. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12842–12851. IEEE, 2025b.

Xingchao Liu, Chengyue Gong, and qiang liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *The Eleventh International Conference on Learning Representations*, 2023.

Charles Lovering, Michael Krumdick, Viet Dac Lai, Varshini Reddy, Seth Ebner, Nilesh Kumar, Rik Koncel-Kedziorski, and Chris Tanner. Language model probabilities are not calibrated in numeric contexts. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29218–29257, 2025.

Aoxin Ni. Numeracy in large language models: Fundamental limitations and paths to improvement. *arXiv preprint arXiv:2608.13129*, 2026.

OpenAI. Introducing gpt-5. `https://openai.com/index/introducing-gpt-5/`, 2025.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition. `https://openai.com/index/gpt-5-6/`, 2026.

Wenqi Ouyang, Yi Dong, Xiaoyang Kang, Peiran Ren, Xin Xu, and Xuansong Xie. Rsfnet: A white-box image retouching approach using region-specific color filters. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 12126–12135. IEEE, 2023.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 4172–4182. IEEE, 2023.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026a. URL `https://qwen.ai/blog?id=qwen3.5`.

Qwen Team. Qwen3.8-Max: A new bar for coding and cowork, August 2026b. URL `https://qwen.ai/blog?id=qwen3.8`.

Yujun Shi, Chuhui Xue, Jun Hao Liew, Jiachun Pan, Hanshu Yan, Wenqing Zhang, Vincent YF Tan, and Song Bai. Dragdiffusion: Harnessing diffusion models for interactive point-based image editing. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8839–8849. IEEE, 2024.

Xingyou Song and Dara Bahri. Decoding-based regression. *arXiv preprint arXiv:2501.19383*, 2025.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, et al. Gemma 4 technical report. *arXiv preprint arXiv:2607.02770*, 2026.

Alexander Tong, Kilian FATRAS, Nikolay Malkin, Guillaume Huguet, Yanlei Zhang, Jarrid Rector-Brooks, Guy Wolf, and Yoshua Bengio. Improving and generalizing flow-based generative models with minibatch optimal transport. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Qiuyue Wang, Mingsheng Li, Jian Guan, Jinhui Ye, Sicheng Xie, Yitao Liu, Junhao Chen, Zhixuan Liang, Jie Zhang, Xintong Hu, et al. Qwen-vla: Unifying vision-language-action modeling across tasks, environments, and robot embodiments. *arXiv preprint arXiv:2605.30280*, 2026a.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025.

Yuqi Wang, Xinghang Li, Wenxuan Wang, Junbo Zhang, Yingyan Li, Yuntao Chen, Xinlong Wang, and Zhaoxiang Zhang. Unified vision-language-action model. In *International Conference on Learning Representations*, volume 2026, pp. 80929–80944, 2026b.

Qiucheng Wu, Yujian Liu, Handong Zhao, Ajinkya Kale, Trung Bui, Tong Yu, Zhe Lin, Yang Zhang, and Shiyu Chang. Uncovering the disentanglement capability in text-to-image diffusion models. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1900–1910. IEEE, 2023.

Qiucheng Wu, Jing Shi, Simon Jenni, Kushal Kafle, Tianyu Wang, Shiyu Chang, and Handong Zhao. Retouchiq: Mllm agents for instruction-based image retouching with generalist reward. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12279–12288, June 2026.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How language model hallucinations can snowball. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 59670–59684. PMLR, 21–27 Jul 2024.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *2018 IEEE/CVF conference on computer vision and pattern recognition*, pp. 586–595. IEEE, 2018.

Kaiwen Zheng, Huayu Chen, Haotian Ye, Haoxiang Wang, Qinsheng Zhang, Kai Jiang, Hang Su, Stefano Ermon, Jun Zhu, and Ming-Yu Liu. Diffusionnft: Online diffusion reinforcement with forward process. In *International Conference on Learning Representations*, volume 2026, pp. 134129–134150, 2026.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023. URL `https://proceedings.mlr.press/v229/zitkovich23a.html`.

A Implementation Details

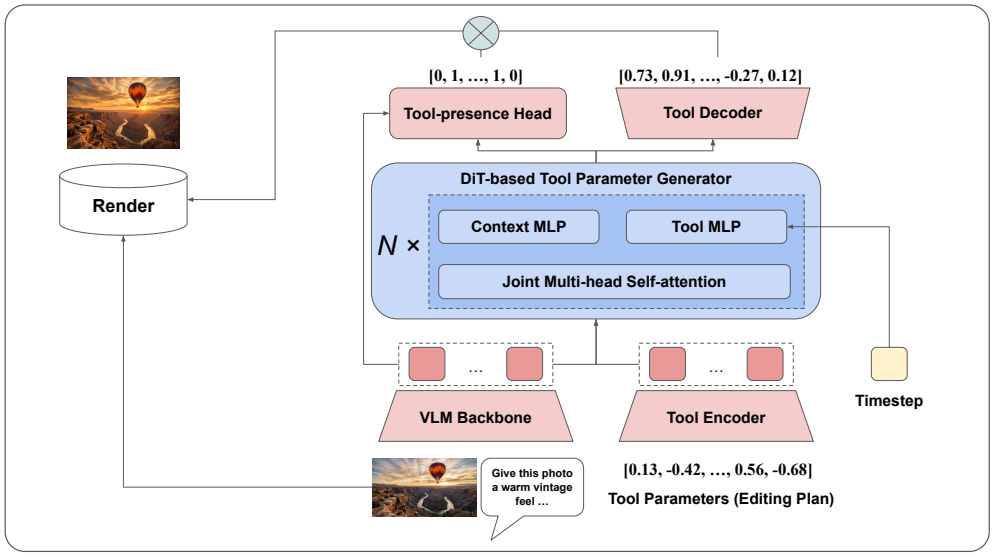

Figure 5: **Overview of the FlowTool architecture.** A vision-language model (VLM) encodes the input image and editing instruction into multimodal conditioning features, while a DiT-based tool parameter generator transforms noisy tool parameters into the final editing plan through conditional rectified flow. The resulting tool parameters are then executed by the image-editing engine.

Compute. Supervised flow-matching training was conducted on a single NVIDIA A100 80GB GPU for approximately 46 hours. Reward-based post-training was performed on a single node with 8 NVIDIA A100 80GB GPUs for approximately 40 hours.

Model Architecture. We use Qwen3.5 (0.8B, 4B, 9B) as the VLM backbone in our experiments. The DiT-based tool parameter generator uses transformer blocks, as in Wang et al. (2026a). The VLM is operated in bfloat16 precision.

B Limitations & Future Works

FlowTool is currently designed for standardized linear parameter spaces, such as parameters normalized to $[-100, 100]$, extending the framework to other parameter structures remains an important direction for future work. In particular, many image-editing and generation systems involve cyclic or polar-valued parameters, such as hue, as well as discrete or categorical parameters, such as textual options. Adapting the model to handle these heterogeneous parameter spaces could further broaden its applicability to more general editing tools and interfaces.