SPARSEEDIT: STAGE-AWARE SPARSE INFERENCE FOR DIT-BASED IMAGE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

Instruction-based image editing with Diffusion Transformers repeatedly processes generated-latent and reference-image latent tokens at every denoising step, leading to substantial inference cost. We study how to organize sparse update patches over denoising under an explicit per-step update budget. A diagnostic based on velocity differences computed from full forward passes shows a transition from larger and more connected update patches early in denoising to increasingly fragmented masks later. This observation motivates SparseEdit, a training-free sparse inference method centered on stage-dependent spatial selection. Its Patch Selector shifts from priorities averaged over neighborhoods to pointwise priorities as denoising progresses while maintaining an exact update budget. A short dense warmup assigns request-level update and velocity-reuse budgets, and an Execution Controller realizes the selected updates through sparse computation, dense refreshes, and velocity reuse. Across GEdit-Bench and KontextBench, SparseEdit achieves 1.83–$2.56\times$ end-to-end speedups on FLUX.1 Kontext Dev, LongCat-Image-Edit, and Qwen-Image-Edit, with PSNR 30.16–35.17 dB, SSIM 0.926–0.992, and LPIPS 0.029–0.051 relative to dense outputs.

1 INTRODUCTION

Instruction-based image editing (IIE) generates an edited image conditioned on a source image and a natural-language instruction. Diffusion Transformer (DiT)-based editors such as FLUX.1 Kontext, Qwen-Image-Edit, and LongCat-Image-Edit encode the source image as reference-image latent tokens, concatenate them with generated-latent tokens, and process both through Transformer blocks at each denoising step (Wu et al., 2025; Meituan LongCat Team et al., 2025; Black Forest Labs et al., 2025). This concatenation nearly doubles the sequence length compared to text-to-image generation, increasing attention and feed-forward computation per step.

Efficient diffusion inference has been studied through few-step methods (Luo et al., 2023; Sauer et al., 2024), pruning (Fang et al., 2023; Castells et al., 2024), quantization (Shang et al., 2023; Li et al., 2025), inter-timestep feature reuse (Selvaraju et al., 2024; Liu et al., 2024), and token-wise sparse computation (Zou et al., 2024; Lou et al., 2024). For image editing, RegionE partitions edited and unedited regions from an early estimate (Chen et al., 2025), while SpotEdit routes tokens based on source similarity (Qin et al., 2025). These methods leave open how the spatial organization of selected update positions should change across denoising stages.

To answer this question, we analyze denoising trajectories using the offline oracle shown in Figure 1, which selects a fixed number of generated-latent positions based on changes in predicted velocity between adjacent steps. The oracle obtains these changes after full forward passes and records the resulting fixed-budget update mask. Each connected component of this mask on the packed latent grid is an update patch. Figure 1 shows a representative trajectory where updating latent velocity only within the selected update patches at a fixed budget of 15% closely reproduces the dense output with an LPIPS of 0.010, indicating that recomputation can be spatially concentrated. The oracle masks contain larger, more connected update patches early in denoising and progressively smaller, more dispersed update patches later. We observe this early-to-late fragmentation pattern consistently across different edit types and multiple DiT-based editors, suggesting that stronger neighborhood aggregation should be used early and gradually reduced as denoising progresses.

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.

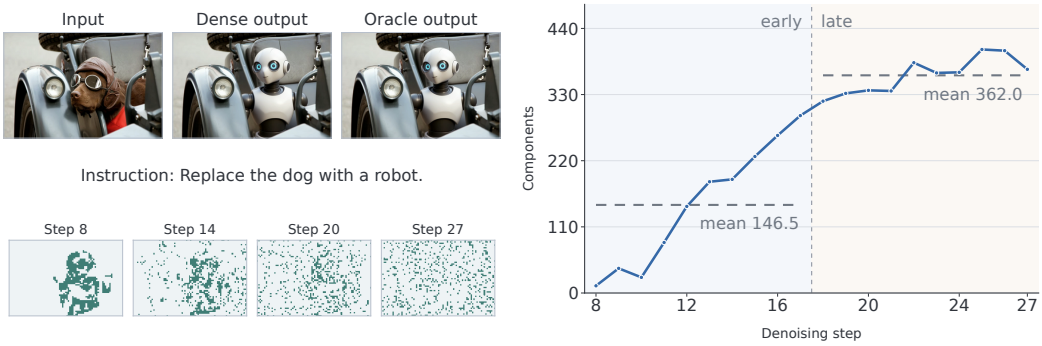


Figure 1: Oracle update masks and spatial connectivity for a representative subject-replacement edit. Left: Input, oracle output (LPIPS 0.010 vs. dense baseline), and instruction. Update masks at selected denoising steps show patches transitioning from large connected regions (teal) to dispersed fragments. Right: Connected-component count increases across denoising, showing progressive spatial fragmentation.

Based on this observation, we propose SparseEdit, a training-free method centered on stage-dependent spatial selection for online sparse inference. The Patch Selector constructs update patches by ranking generated-latent positions using recent velocity change and token age, then applying a locality prior that weakens over denoising under an exact update budget. The sparse forward pass recomputes only the selected positions, with active queries that attend to the complete context of cached generated-latent tokens, reference-image latent tokens, and multimodal conditioning tokens. For each request, Warmup-Based Policy Routing runs a brief warmup that measures late change magnitude, convergence ratio, and estimated edit coverage, and maps the request to an execution profile specifying the update ratio and velocity-reuse budget. For each remaining step, the Execution Controller chooses among refresh, sparse computation, and velocity reuse, which skips the forward pass by scaling the previously used velocity.

In this paper, we make the following contributions:

- Using a velocity-difference oracle with a fixed budget, we analyze the spatial organization of update patches across denoising. Update patches are larger and more connected early in denoising and progressively more fragmented later. This early-to-late fragmentation pattern occurs consistently across different edit types and multiple DiT-based editors, motivating a stage-dependent spatial bias for sparse selection.
- We introduce SparseEdit, a training-free sparse inference method centered on stage-dependent spatial selection. Its Patch Selector shifts from priorities averaged over neighborhoods to pointwise priorities while maintaining an exact update budget at each sparse computation step. Warmup-Based Policy Routing assigns request-level update and reuse budgets, and the Execution Controller realizes the selected updates through cached sparse computation, dense refreshes, and velocity reuse.
- Across GEdit-Bench and KontextBench, our vLLM-Omni implementation of SparseEdit achieves 2.28–$2.56\times$, 1.83–$2.21\times$, and 2.10–$2.24\times$ end-to-end speedups on FLUX.1 Kontext Dev, LongCat-Image-Edit, and Qwen-Image-Edit, respectively, with PSNR 30.16–35.17 dB, SSIM 0.926–0.992, and LPIPS 0.029–0.051 relative to dense outputs.

2 Background and Related Work

Instruction-based image editing (IIE) with DiTs repeatedly processes generated-latent, reference-image latent, and multimodal conditioning tokens across denoising steps (Wu et al., 2025; Meituan LongCat Team et al., 2025; Black Forest Labs et al., 2025; Liu et al., 2025b). Existing acceleration methods for diffusion models can be grouped into inter-timestep feature reuse, token-wise feature caching, and region-aware acceleration for image editing. However, none of these methods jointly address per-request computation budgeting, stage-dependent spatial organization of update patches,

and adaptive velocity reuse under an explicit per-step budget constraint. We review each direction below and highlight the gaps that motivate SparseEdit.

Inter-Timestep Feature Reuse Inter-timestep methods exploit similarity between nearby denoising states. FORA reuses features at fixed intervals, TeaCache triggers reuse from output-change estimates, Δ-DiT caches stage-dependent block offsets, and TaylorSeer extrapolates features from temporal derivatives (Selvaraju et al., 2024; Liu et al., 2024; Chen et al., 2024; Liu et al., 2025a). Cache-DiT supports block-level feature reuse through its DBCache configuration (Cache-DiT Contributors, 2025). Their decisions generally apply to a whole block, module, or sampling step, so all input tokens share the same action. This treatment overlooks spatially non-uniform update demand in IIE and does not track the age of each token's cached state or adapt reuse aggressiveness to editing difficulty. It therefore leaves where and how much computation to allocate within each step unresolved.

Token-Wise Feature Caching Token-wise methods allocate computation non-uniformly across input tokens. RAS uses previous-step noise statistics, ToCa combines attention and cache scores, TokenCache learns caching decisions, and DaTo reconstructs pruned tokens from retained features (Liu et al., 2025c; Zou et al., 2024; Lou et al., 2024; Zhang et al., 2024). Existing methods rank tokens using token-level metrics individually and determine the number of update tokens through a global ratio schedule. ToCa enforces spatial uniformity in token selection to prevent error accumulation, while our work investigates whether the spatial aggregation bias of selected tokens should vary across denoising stages. These signals are primarily designed for text-to-image denoising and are not explicitly calibrated for source-conditioned editing, where the computation demand depends on both the source image and the edit instruction. They therefore do not directly address our setting, which requires assigning a request-level update budget and adapting the spatial organization of update patches over the denoising process.

Region-Aware Acceleration for Image Editing Region-aware methods identify reusable content through edit masks or source-image comparison. EEdit refreshes tokens in an edit-local region and its neighborhood while reusing cached features elsewhere, RegionE combines an early edited/unedited partition with local denoising, KV reuse, and velocity reuse, and SpotEdit retains cached attention context for regions skipped according to source similarity (Yan et al., 2025; Chen et al., 2025; Qin et al., 2025). These policies perform well in localized, source-preserving edits. However, edit regions vary across requests, thus the fraction of tokens that these methods can skip or reuse is also request dependent. Their effective token-update ratio and speedup therefore vary across tasks and may decrease for global edits, where little content remains close to the source image. These limitations motivate a sparse update method with an explicit computation budget for different requests and a stage-dependent organization of selected tokens into patches.

3 PRELIMINARIES

Image-edit token groups and iterative denoising We use Qwen-Image-Edit as a representative DiT-based editor (Wu et al., 2025). Its image stream mainly contains evolving generated-latent tokens Z_i and reference-image latent tokens R. Let $N = |Z_i|$ denote the number of generated-latent tokens. The initial sequence Z_0 is formed by packing Gaussian-noise latents, while the source image is VAE-encoded and packed into R. Let $\tau_i \in [0, 1]$ denote the normalized flow time, which decreases from $\tau_0 = 1$ toward the clean endpoint $\tau_T = 0$ as denoising progresses, and t_i the corresponding model timestep. At denoising step i, with the edit instruction implicit in the notation,

$$\begin{aligned} V_i &= \mathrm{DiT}([Z_i; R], t_i)_{1:N}, \\ Z_{i+1} &= Z_i + (\tau_{i+1} - \tau_i) V_i. \end{aligned} \tag{1}$$

The DiT processes the concatenated sequence $[Z_i; R]$, producing velocity predictions for every position in the image token sequence. The notation $(\cdot)_{1:N}$ extracts the first N rows corresponding to the generated-latent tokens. The scheduler updates Z_i using these velocity predictions, while the reference input R remains fixed. Its hidden states and attention projections can still change across denoising steps.

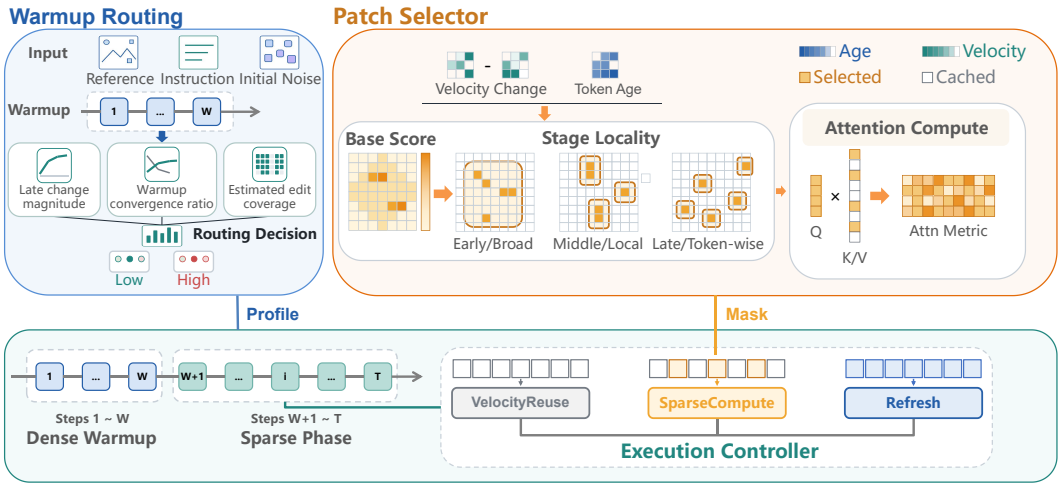


Figure 2: Overview of SparseEdit. Warmup-Based Policy Routing runs W denoising steps to measure trajectory characteristics and select an execution profile specifying update budget K and velocity-reuse budget B. The Stage-Aware Patch Selector ranks generated-latent positions using recent velocity change and token age, then interpolates between locally averaged and pointwise priorities based on denoising progress to construct update mask M_i with $\|M_i\|_0 = K$. The Execution Controller orchestrates three actions: REFRESH performs dense computation, VELOCITYREUSE extrapolates previous predictions based on B, and SPARSECOMPUTE updates selected positions under M_i while retaining cached states for inactive tokens.

Offline velocity-difference oracle. Let $V_i(p)$ denote the velocity prediction at position p. Following the terminology in Section 1, spatially connected selected positions form an update patch. To identify which update patches require new velocity predictions, the offline oracle in Section 1 uses outputs from full forward passes at adjacent denoising steps. It measures the velocity change at each position, selects the top-K positions with largest changes, and replaces their retained velocities with current predictions. We use tildes ($\widetilde{V}$) to denote retained or blended velocity fields, distinguishing them from fresh predictions (V):

$$\begin{aligned} D_i(p) &= \|V_i(p) - V_{i-1}(p)\|_2, \\ \widetilde{V}_i^{\mathrm{orc}}(p) &= M_i^{\mathrm{orc}}(p) V_i(p) + \big(1 - M_i^{\mathrm{orc}}(p)\big) \widetilde{V}_{i-1}^{\mathrm{orc}}(p), \\ &\text{where } M_i^{\mathrm{orc}} = \mathrm{TopKMask}(D_i, K). \end{aligned} \tag{2}$$

The retained velocity field is initialized with $\widetilde{V}_0^{\mathrm{orc}} = V_0$ and is used in place of V_i for the scheduler update. Online sparse inference must select update patches from states available before full denoising and maintain both cached Transformer states and retained velocity predictions across subsequent steps.

4 METHODOLOGY

SparseEdit is centered on a stage-dependent spatial selection rule. Given a request-level update budget, the Patch Selector gradually reduces the influence of neighborhood averaging on the priorities as denoising progresses, shifting from locally grouped update patches toward more spatially dispersed updates while maintaining an exact budget. Warmup-Based Policy Routing assigns the request's update and velocity-reuse budgets, and the Execution Controller realizes the selected updates through cached sparse computation, dense refreshes, and velocity reuse. Figure 2 gives an overview of the full pipeline, and we describe its components in turn.

Auxiliary conditioning sequences, such as text tokens, are fully computed and updated in every Transformer forward pass of SparseEdit. They are excluded from patch selection and the generated-token update budget, and their processing is left implicit in the following description.

4.1 WARMUP-BASED POLICY ROUTING

Warmup runs W full denoising steps to initialize Transformer caches, measure trajectory characteristics, and select an execution profile that provides update budget K and velocity-reuse budget B. Let $\bar{d}_i^{\text{clean}}$ denote the mean per-token distance between consecutive clean predictions $Z_i - \tau_i V_i$ at transition $i-1 \to i$, and $d_{\text{ref}}(p)$ the distance between the final warmup prediction and reference latent at p. Using the first and last two warmup transitions and threshold η_{edit}, warmup computes

$$\begin{aligned} m_{\text{late}} &= \frac{\bar{d}_{W-2}^{\text{clean}} + \bar{d}_{W-1}^{\text{clean}}}{2}, \\ m_{\text{conv}} &= \frac{m_{\text{late}}}{(\bar{d}_1^{\text{clean}} + \bar{d}_2^{\text{clean}})/2 + \epsilon}, \\ m_{\text{edit}} &= \frac{1}{N} \sum_{p=1}^{N} \mathbf{1}[d_{\text{ref}}(p) > \eta_{\text{edit}}]\,. \end{aligned} \tag{3}$$

Here, m_{late} measures late prediction change magnitude, m_{conv} compares late change with early change, and m_{edit} estimates the fraction of tokens departing from the reference. These metrics are combined via equal weighting into a routing score compared against a threshold θ_{route} to select between two predefined execution profiles. An aggressive profile with smaller K and larger B is used for stable trajectories and narrow edits, while a conservative profile with higher K and smaller B is used otherwise.

4.2 STAGE-AWARE PATCH SELECTOR

Given the profile's update budget K, the Patch Selector constructs spatially coherent update patches in early denoising steps and gradually transitions to dispersed corrections as denoising progresses. Early steps require contiguous regional updates to establish structural coherence, while later steps demand precise, scattered refinements to resolve local inconsistencies. The stage-aware transition reconciles these competing demands while maintaining a strict per-step computation budget.

Selection base metrics. At step i, the velocity change $d_i(p) = \|\widetilde{V}_{i-1}(p) - \widetilde{V}_{i-2}(p)\|_2$ compares consecutive retained predictions available before the current forward pass. Token age $a_i(p)$ counts steps since last computation at p, reset after each computation. We normalize $d_i(p)(1 + 0.5a_i(p))$ across generated positions to form $s_i(p)$, retaining the base signal when age is zero.

Locality transition mechanism. To implement stage-dependent spatial coherence, we interpolate between locally averaged and base scores. Let $\text{Avg}_{5\times5}(s_i)(p)$ denote the 5×5 neighborhood mean at p, inducing soft patch grouping through overlapping windows.

Let τ_W denote the normalized flow time at the end of warmup (step W). The interpolated priority map and mask construction are

$$\begin{aligned} \bar{s}_i(p) &= \frac{\tau_W - \tau_i}{\tau_W}\, s_i(p) + \frac{\tau_i}{\tau_W}\, \text{Avg}_{5\times5}(s_i)(p), \\ M_i &= \text{TopKMask}(\bar{s}_i, K), \qquad \|M_i\|_0 = K. \end{aligned} \tag{4}$$

The linear interpolation shifts from purely averaged scores at $\tau_i = \tau_W$ (maximizing coherence) to direct base scores as $\tau_i \to 0$ (permitting dispersed updates), mirroring oracle progression. Fixed locality bias produces over-smoothed edits, as spatial coherence persists into late steps where targeted corrections are needed. Fixed dispersed selection fragments early structure, preventing coherent region formation.

Budget enforcement. The TopKMask operation selects exactly the K highest-priority positions from $\bar{s}_i$, producing a binary mask M_i with $\|M_i\|_0 = K$. The selected mask is passed to the Execution Controller, which applies M_i during SPARSECOMPUTE to determine which generated-latent states and velocity predictions to recompute and which token ages to reset.

4.3 EXECUTION CONTROLLER

The Execution Controller orchestrates three actions. REFRESH performs dense computation over all tokens, VELOCITYREUSE extrapolates the previous velocity prediction, and SPARSECOMPUTE performs partial computation over K selected positions. It maintains a consumable reuse budget $\mathcal{B}_i$ to regulate when velocity extrapolation remains sufficiently accurate. When the budget permits, the controller accepts VELOCITYREUSE. Otherwise it invokes the Patch Selector for SPARSECOMPUTE. The selected profile supplies initial budget B, with $\mathcal{B}_W = B$ after warmup. Dense refresh is invoked at fixed intervals to prevent accumulated approximation errors from consecutive reuse and sparse steps.

Reuse score. Let $\widetilde{V}_i$ denote the complete velocity field used by the scheduler at step i, following the notation in Eq. 1. This field may come from fresh computation, velocity reuse, or sparse computation that blends new and cached predictions. At step $i \geq 2$, the reuse score g_i estimates the deviation associated with reusing the previous velocity instead of computing a fresh prediction. When the preceding step computed new states, the score compares consecutive velocity predictions, scales their difference by the current scheduler step size, and normalizes it by the current latent magnitude. Here r_i denotes a timestep-aware scaling factor that adjusts velocity predictions across denoising steps according to the sampling schedule's local step size:

$$g_i = \frac{|\tau_{i+1} - \tau_i|}{\|Z_i\|_F + \epsilon} \left\| \widetilde{V}_{i-1} - r_{i-1}\widetilde{V}_{i-2} \right\|_F \tag{5}$$

Smaller scores indicate stable velocity fields suitable for reuse, while larger scores signal instability and consume more budget when reuse is accepted. When the preceding step itself reused velocity rather than computing, we extrapolate the score using a fixed penalty factor α as $g_i = \alpha g_{i-1}$ to reflect increased uncertainty from consecutive reuse.

Action selection and velocity formation. When $g_i \leq \mathcal{B}_i$, the controller accepts VELOCITYREUSE and extrapolates $\widetilde{V}_i = r_i\widetilde{V}_{i-1}$, consuming budget according to $\mathcal{B}_{i+1} = \mathcal{B}_i - g_i$. Otherwise it invokes the Patch Selector for SPARSECOMPUTE. Each computed step restores budget to B.

Let V_i denote fresh velocity predictions from the current forward pass, and $\widehat{V}_i(p)$ the most recent computed prediction retained at position p. With M_i broadcast over velocity channels, the scheduler velocity is formed as

$$\widetilde{V}_i = \begin{cases} V_i, & \text{REFRESH}, \\ r_i\widetilde{V}_{i-1}, & \text{VELOCITYREUSE}, \\ M_i \odot V_i + (1 - M_i) \odot \widehat{V}_i, & \text{SPARSECOMPUTE}. \end{cases} \tag{6}$$

Refresh resets all token ages to zero, sparse computation resets only active ages while incrementing inactive ages, and reuse increments every age. Accumulated ages influence the next Patch Selector invocation.

Sparse attention and cache updates. During SPARSECOMPUTE, mask M_i selects K active positions. Active queries attend to full cached context, which consists of fresh projections at active positions, cached projections at inactive positions, and cached reference-image latent tokens. At block ℓ,

$$\mathbf{O}^{(\ell)}_{i,\mathcal{A}_i} = \mathrm{Softmax}\left(\frac{\mathbf{Q}^{(\ell)}_{i,\mathcal{A}_i} (\mathbf{K}^{(\ell)}_i)^\top}{\sqrt{d_h}} \right) \mathbf{V}^{(\ell)}_i, \tag{7}$$

where $\mathcal{A}_i = \{p : M_i(p) = 1\}$ denotes active positions, $\mathbf{Q}^{(\ell)}_{i,\mathcal{A}_i}$ contains active queries, and $\mathbf{K}^{(\ell)}_i, \mathbf{V}^{(\ell)}_i$ contain all keys and values across generated-latent and reference-image latent positions. The active attention outputs pass through the output projection, residual connection, and feed-forward network. New active key-value pairs replace their cache entries while inactive cache entries remain unchanged.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

Models and datasets. We evaluate SparseEdit on three open-source image editors: FLUX.1 Kontext Dev (Black Forest Labs et al., 2025), LongCat-Image-Edit (Meituan LongCat Team et al., 2025), and Qwen-Image-Edit (Wu et al., 2025). Each editor is evaluated on GEdit-Bench (Liu et al., 2025b) (606 English image-instruction pairs across 11 tasks) and KontextBench (Black Forest Labs et al., 2025) (1,026 pairs across five tasks). All main experiments use 28 denoising steps with classifier-free guidance (CFG). Qwen uses CFG scale 4.0 and guidance scale 1.0; FLUX uses true CFG scale 4.0 and distilled guidance 3.5. LongCat uses guidance scale 3.5 to enable CFG (> 1), with its separate CFG-scale parameter set to 1.0. Negative prompts are a single space for Qwen and FLUX and empty for LongCat.

Metrics. Following RegionE (Chen et al., 2025), we evaluate the quality–efficiency trade-off using dense outputs as the fidelity reference. PSNR (↑), SSIM (Wang et al., 2004) (↑), and LPIPS (Zhang et al., 2018) (↓) measure pixel-level fidelity, structural similarity, and perceptual discrepancy, respectively, providing complementary views of changes introduced by acceleration. Paired runs use the same editor, source image, instruction, seed, and guidance settings. Efficiency is measured by mean single-request end-to-end latency and speedup relative to the paired dense run.

Baselines. We select representative baselines covering temporal reuse, token-wise caching, and region-aware editing to compare SparseEdit with both general diffusion acceleration and editing-specific methods. TeaCache (Liu et al., 2024) and Cache-DiT (Cache-DiT Contributors, 2025) reuse features across denoising steps to exploit temporal redundancy. RAS (Liu et al., 2025c) and ToCa (Zou et al., 2024) selectively update tokens to reduce spatial computation. RegionE (Chen et al., 2025) and SpotEdit (Qin et al., 2025) focus computation on edit-relevant regions, providing direct comparisons with methods designed for image editing.

Implementation details. SparseEdit is implemented in vLLM-Omni with eight warmup steps and a dense refresh interval of 15 steps. We set $\eta_{\text{edit}} = 3.0$, $\theta_{\text{route}} = 0.50$, equal routing weights of $1/3$, and reuse-score penalty $\alpha = 1.5$. The aggressive and conservative profiles use $(K, B) = (0.20N, 0.02)$ and $(0.40N, 0.01)$, respectively. All main experiments run on a single NVIDIA H20 GPU with batch size 1, editing one image per request and running one model at a time.

5.2 OVERALL QUALITY–EFFICIENCY COMPARISON

Observations. On GEdit-Bench, SparseEdit achieves 2.560× speedup on FLUX.1 Kontext Dev with LPIPS 0.029, PSNR 35.168 dB, and SSIM 0.955, outperforming the compared baselines on all four metrics. On the same benchmark, it retains SSIM 0.992 and LPIPS 0.042 at 2.211× speedup on LongCat-Image-Edit. Across all six evaluated combinations of model and benchmark, SparseEdit achieves the lowest LPIPS and highest speedup, shown in Table 1. This trade-off is consistent with its joint spatial and temporal adaptation. Relative to inter-timestep feature reuse (Cache-DiT, TeaCache), spatial selection allows stable tokens to remain cached while regions requiring updates are recomputed. Relative to token-wise feature caching (RAS, ToCa), stage-aware selection shifts from coherent early patches to dispersed late corrections, supporting structural coherence and detail refinement. For region-aware methods (RegionE, SpotEdit), the available reuse depends on edit locality; SparseEdit enforces an explicit per-request update budget, reducing reliance on large source-preserving regions. Warmup routing jointly selects the spatial update budget K and velocity-reuse budget B according to edit difficulty, while the execution controller skips Transformer evaluations when the reuse criterion is satisfied.

Visualization. Figure 3 presents qualitative comparisons on four editing tasks using FLUX.1 Kontext Dev. The visualization demonstrates that SparseEdit preserves fine-grained details such as complete object structure, legible text, background coherence, and subject identity, while other acceleration methods introduce artifacts including incomplete objects, blurred lettering, and loss of structural detail. These results confirm that from a visual quality perspective, SparseEdit consistently produces edits closest to the dense reference.

Across three image editors and two benchmarks, SparseEdit achieves the highest speedup and lowest LPIPS among the compared methods, while the qualitative examples illustrate its preservation

Table 1: Quality-efficiency comparison across three pretrained image editors. Speedup is normalized to the corresponding paired dense run. IFR, TFC, and RA denote inter-timestep feature reuse, token-wise feature caching, and region-aware acceleration for image editing, respectively.

	GEdit-Bench					KontextBench				
Method	LPIPS ↓	PSNR ↑	SSIM ↑	Latency(s) ↓	Speedup ↑	LPIPS ↓	PSNR ↑	SSIM ↑	Latency(s) ↓	Speedup ↑
LongCat-Image-Edit										
Dense	–	–	–	49.19	1.000×	–	–	–	49.16	1.000×
Cache-DiT IFR	0.064	28.302	0.991	28.45	1.729×	0.082	24.403	0.909	28.40	1.731×
TeaCache IFR	0.049	28.785	0.978	23.35	2.107×	0.061	26.792	0.917	29.94	1.642×
RAS TFC	0.070	26.990	0.981	32.11	1.532×	0.075	24.455	0.897	32.93	1.493×
ToCa TFC	0.052	30.471	0.977	36.06	1.364×	0.079	24.394	0.905	40.39	1.217×
SpotEdit RA	0.056	28.712	0.984	32.23	1.526×	0.052	29.212	0.926	37.82	1.300×
RegionE RA+IFR	0.048	30.078	0.991	26.78	1.837×	0.058	27.086	0.917	30.35	1.620×
SparseEdit RA+IFR	0.042	31.103	0.992	22.25	2.211×	0.050	30.342	0.933	26.91	1.827×
FLUX.1 Kontext Dev										
Dense	–	–	–	83.76	1.000×	–	–	–	83.77	1.000×
Cache-DiT IFR	0.048	32.852	0.934	36.06	2.323×	0.055	32.575	0.929	43.93	1.907×
TeaCache IFR	0.077	28.125	0.924	35.86	2.336×	0.082	28.028	0.878	45.48	1.842×
RAS TFC	0.054	33.524	0.930	47.78	1.753×	0.094	28.998	0.868	50.34	1.664×
ToCa TFC	0.074	29.803	0.906	60.61	1.382×	0.062	29.638	0.898	64.29	1.303×
SpotEdit RA	0.052	33.621	0.937	43.99	1.904×	0.063	29.387	0.913	49.86	1.680×
RegionE RA+IFR	0.059	34.327	0.947	36.28	2.309×	0.058	31.215	0.913	39.59	2.116×
SparseEdit RA+IFR	0.029	35.168	0.955	32.72	2.560×	0.051	33.221	0.948	36.81	2.276×
Qwen-Image-Edit										
Dense	–	–	–	109.85	1.000×	–	–	–	109.91	1.000×
Cache-DiT IFR	0.097	30.299	0.901	57.69	1.904×	0.050	29.897	0.916	72.36	1.519×
TeaCache IFR	0.050	31.766	0.969	57.06	1.925×	0.067	27.917	0.917	57.70	1.905×
RAS TFC	0.073	25.727	0.910	73.92	1.486×	0.052	28.716	0.924	81.23	1.353×
ToCa TFC	0.075	25.956	0.919	81.13	1.354×	0.068	26.408	0.909	85.93	1.279×
SpotEdit RA	0.042	29.708	0.976	68.31	1.608×	0.097	23.854	0.876	63.90	1.720×
RegionE RA+IFR	0.041	29.355	0.979	49.57	2.216×	0.061	27.825	0.928	54.93	2.001×
SparseEdit RA+IFR	0.037	31.742	0.981	49.13	2.236×	0.049	30.158	0.926	52.36	2.099×

of structure and fine details. These results are consistent with its stage-aware selection of coherent update patches early in denoising and dispersed corrections later, complemented by request-adaptive budgets and velocity reuse. Together, the findings support adapting sparse computation to the evolving spatial requirements of denoising as an effective approach to accelerating DiT-based image editing.

5.3 Ablation Studies

We ablate stage-aware locality, velocity reuse, and warmup routing on Qwen-Image-Edit using the full GEdit-Bench dataset in Table 2.

Stage-aware locality. Table 2 demonstrates that stage-aware priority modeling is essential for maintaining output fidelity while achieving acceleration. Without stage-aware adaptation, locality scores computed using fixed spatial aggregation (5×5 average pooling) degrade LPIPS to 0.064, while fixed pointwise raw scores yield 0.058 LPIPS, despite maintaining nearly identical speedup. These results confirm that stage-aware priority modeling enables the spatial importance estimator to adapt token selection criteria as denoising progresses, identifying edit-relevant regions more accurately in early refinement stages and preserving detail fidelity in later convergence stages. The substantial quality improvement at constant computational cost demonstrates that temporal context is critical for spatial sparsity decisions in diffusion-based image editing.

Velocity reuse and warmup routing. Table 2 demonstrates that velocity reuse and warmup routing contribute through complementary mechanisms. Velocity reuse achieves substantial acceleration at minimal perceptual cost by skipping Transformer evaluations when denoising trajectories are stable. Enabling velocity reuse increases speedup from $1.958\times$ to $2.236\times$ while degrading LPIPS by only 0.006, which shows that velocity reuse delivers 14% throughput gains while maintaining output quality. Warmup routing enables per-request profile selection that balances quality and efficiency according to edit complexity. Using the conservative profile uniformly yields low speedup at $1.710\times$ despite strong quality at 0.025 LPIPS, while using the aggressive profile uniformly achieves $2.537\times$ speedup but suffers notable quality degradation at 0.060 LPIPS. SparseEdit's warmup routing adaptively selects between these profiles based on measured edit difficulty, reaching $2.236\times$ speedup at 0.037 LPIPS by matching both spatial and temporal budgets to request complexity. Each profile specifies both the spatial update budget K and the velocity-reuse budget B, enabling joint optimiza-

Table 2: Ablation studies of stage-aware locality, velocity reuse and warmup routing. SparseEdit combines all these components.

Configuration	LPIPS ↓	PSNR ↑	SSIM ↑	Δ LPIPS	Speedup
SparseEdit	0.037	31.742	0.981	–	2.236×
Stage-Aware Locality Ablation					
Fixed Locality (5×5 average)	0.064	28.865	0.908	+0.027	2.236×
Fixed Pointwise (raw scores)	0.058	30.021	0.896	+0.021	2.239×
Velocity Reuse Ablation					
Without velocity reuse	0.031	32.953	0.989	−0.006	1.958×
Warmup Routing Ablation					
Fixed Aggressive Routing	0.060	30.862	0.928	+0.023	2.537×
Fixed Conservative Routing	0.025	38.364	0.986	−0.012	1.710×



Figure 3: Editing examples on FLUX.1 Kontext Dev By SparseEdit and other baselines.

tion across spatial token selection and temporal step skipping. These two ablation axes confirm that velocity reuse optimizes temporally by skipping stable steps, whereas warmup routing jointly configures spatial and temporal optimization parameters through adaptive profile selection.

6 Conclusion

We studied how to allocate sparse update patches in DiT-based image editing under an exact per-step update budget. A diagnostic based on velocity differences under a fixed update budget revealed that update patches shift from locally concentrated early in denoising to spatially fragmented later. SparseEdit turns this stage-dependent behavior into an online selection rule whose locality bias weakens over denoising while preserving the update budget. Warmup-Based Policy Routing adapts budgets to each request, and the Execution Controller realizes selected patches through cached sparse computation and velocity reuse. Together, these mechanisms provide a practical execution path for stage-aware spatial selection without additional training, approximating dense inference through sparse updates and cached states.

REPRODUCIBILITY STATEMENT

We provide the necessary code, datasets, evaluation scripts, and the raw data used to derive our experimental conclusions in the supplementary material to ensure reproducibility.

AI USE STATEMENT

We used generative AI tools to improve the clarity, grammar, and readability of portions of the author-written manuscript. The authors reviewed and revised all AI-assisted text and verified the final technical claims, citations, and reported results. We take full responsibility for the content of this paper.

REFERENCES

Black Forest Labs, Stephen Batifol, Andreas Blattmann, Frederic Boesel, Saksham Consul, Cyril Diagne, Tim Dockhorn, Jack English, Zion English, Patrick Esser, et al. Flux.1 kontext: Flow matching for in-context image generation and editing in latent space. *arXiv preprint arXiv:2506.15742*, 2025.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. Dicache: Let diffusion model determine its own cache. *arXiv preprint arXiv:2508.17356*, 2025.

Cache-DiT Contributors. Cache-dit: A pytorch-native inference engine with cache, parallelism, quantization and cpu offload for dits. `https://github.com/vipshop/cache-dit`, 2025.

Thibault Castells, Hyoung-Kyu Song, Bo-Kyeong Kim, and Shinkook Choi. Ld-pruner: Efficient pruning of latent diffusion models using task-agnostic insights. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2024.

Pengtao Chen, Mingzhu Shen, Peng Ye, Jianjian Cao, Chongjun Tu, Christos-Savvas Bouganis, Yiren Zhao, and Tao Chen. Δ-DiT: A training-free acceleration method tailored for diffusion transformers. *arXiv preprint arXiv:2406.01125*, 2024.

Pengtao Chen, Xianfang Zeng, Maosen Zhao, Mingzhu Shen, Peng Ye, Bangyin Xiang, Zhibo Wang, Wei Cheng, Gang Yu, and Tao Chen. Regione: Adaptive region-aware generation for efficient image editing. *arXiv preprint arXiv:2510.25590*, 2025.

Gongfan Fang, Xinyin Ma, and Xinchao Wang. Structural pruning for diffusion models. In *Advances in Neural Information Processing Systems*, 2023.

Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *International Conference on Learning Representations*, 2025.

Feng Liu, Shiwei Zhang, Xiaofeng Wang, Yujie Wei, Haonan Qiu, Yuzhong Zhao, Yingya Zhang, Qixiang Ye, and Fang Wan. Timestep embedding tells: It's time to cache for video diffusion model. *arXiv preprint arXiv:2411.19108*, 2024.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15853–15863, Honolulu, HI, USA, 2025a. IEEE. doi: 10.1109/ICCV51701.2025.01471.

Shiyu Liu, Yucheng Han, Peng Xing, Fukun Yin, Rui Wang, Wei Cheng, Jiaqi Liao, Yingming Wang, Honghao Fu, Chunrui Han, Guopeng Li, Yuang Peng, Quan Sun, Jingwei Wu, Yan Cai, Zheng Ge, Ranchen Ming, Lei Xia, Xianfang Zeng, Yibo Zhu, Binxing Jiao, Xiangyu Zhang, Gang Yu, and Daxin Jiang. Step1x-edit: A practical framework for general image editing. *arXiv preprint arXiv:2504.17761*, 2025b.

Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers. *arXiv preprint arXiv:2502.10389*, 2025c.

Jinming Lou, Wenyang Luo, Yufan Liu, Bing Li, Xinmiao Ding, Weiming Hu, Yuming Li, and Chenguang Ma. Token caching for diffusion transformer acceleration. *arXiv preprint arXiv:2409.18523*, 2024.

Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference. *arXiv preprint arXiv:2310.04378*, 2023.

Meituan LongCat Team, Hanghang Ma, Haoxian Tan, Jiale Huang, Junqiang Wu, Jun-Yan He, Lishuai Gao, Songlin Xiao, Xiaoming Wei, Xiaoqi Ma, Xunliang Cai, Yayong Guan, and Jie Hu. Longcat-image technical report. *arXiv preprint arXiv:2512.07584*, 2025.

Zhibin Qin, Zhenxiong Tan, Zeqing Wang, Songhua Liu, and Xinchao Wang. Spotedit: Selective region editing in diffusion transformers. *arXiv preprint arXiv:2512.22323*, 2025.

Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *European Conference on Computer Vision*, 2024.

Pratheba Selvaraju, Tianyu Ding, Tianyi Chen, Ilya Zharkov, and Luming Liang. Fora: Fast-forward caching in diffusion transformer acceleration. *arXiv preprint arXiv:2407.01425*, 2024.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Chenfei Wu, Jiahao Li, Jingren Zhou, Junyang Lin, Kaiyuan Gao, et al. Qwen-image technical report. *arXiv preprint arXiv:2508.02324*, 2025.

Zexuan Yan, Yue Ma, Chang Zou, Wenteng Chen, Qifeng Chen, and Linfeng Zhang. Eedit: Rethinking the spatial and temporal redundancy for efficient image editing. *arXiv preprint arXiv:2503.10270*, 2025.

Evelyn Zhang, Bang Xiao, Jiayi Tang, Qianli Ma, Chang Zou, Xuefei Ning, Xuming Hu, and Linfeng Zhang. Token pruning for caching better: 9 times acceleration on stable diffusion for free. *arXiv preprint arXiv:2501.00375*, 2024.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

Xin Zhou, Dingkang Liang, Kaijin Chen, Tianrui Feng, Xiwu Chen, Hongkai Lin, Yikang Ding, Feiyang Tan, Hengshuang Zhao, and Xiang Bai. Less is enough: Training-free video diffusion acceleration via runtime-adaptive caching. *arXiv preprint arXiv:2507.02860*, 2025.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. *arXiv preprint arXiv:2410.05317*, 2024.

APPENDIX

A ADDITIONAL ORACLE DIAGNOSTICS ACROSS EDITING TASKS

We complement the single-trajectory diagnostic in Figure 1 with a full-information diagnostic on images paired with instructions for local, global, style, and text edits, to examine whether the early-to-late change in mask connectivity also occurs outside subject replacement.

Oracle Implement Setup. We use a fixed 15% update budget for oracle setup. Inputs are resized with their aspect ratios preserved and centered on a white 1024×1024 canvas. The generated latent has a 64×64 packed-token grid, giving exactly $K = \lfloor 0.15 \times 4096 \rfloor = 614$ selected positions at every step from 8 through 27. Dense and oracle runs use the same model, input, prompt, precision, sampling schedule, and initial noise, with full BF16 weights model on NVIDIA H20 GPUs.

Let $V_i(p)$ denote the velocity prediction at position p, and let spatially connected selected positions form an update patch. The oracle score measures the velocity change between adjacent full forward passes as $D_i(p) = \|V_i(p) - V_{i-1}(p)\|_2$, and the retained velocity field is updated by

$$\widetilde{V}_i^{\mathrm{orc}}(p) = M_i^{\mathrm{orc}}(p) V_i(p) + \big(1 - M_i^{\mathrm{orc}}(p)\big) \widetilde{V}_{i-1}^{\mathrm{orc}}(p), \qquad M_i^{\mathrm{orc}} = \mathrm{TopKMask}(D_i, K), \tag{8}$$

initialized with $\widetilde{V}_0^{\mathrm{orc}} = V_0$ and used in place of V_i for the scheduler update.Warmup initializes the retained field with $\widetilde{V}_i = V_i$, and all Transformer positions are freshly computed at every step, so only the scheduler velocity is selectively updated and there is no sparse attention, velocity decay, or dense refresh. This variant requires full information and does not measure an achievable inference speedup.

Selected visual examples. The following three figures use the same display convention as supplementary proof for oracle update masks and spatial connectivity. Each figure shows the source, dense output, and oracle output, followed by masks at steps 8, 14, 20, and 27. The curve includes every measured sparse step, and dashed lines show the early and late means.

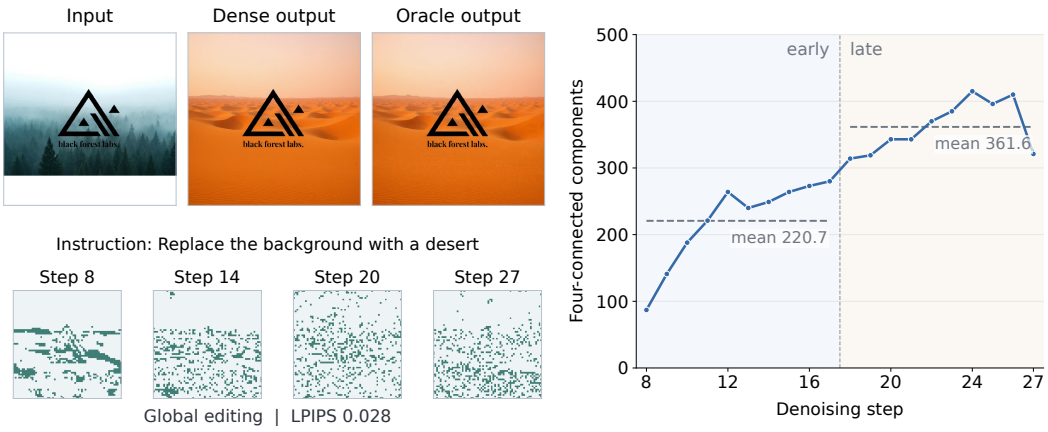


Figure 4: Global editing with FLUX.1 Kontext Dev (KontextBench request `0009_11`). With a fixed 15% budget, the LPIPS between the oracle and dense outputs is 0.028. The mean component count increases from 220.7 to 361.6.

B SPARSEEDIT INFERENCE FORMULATION AND ALGORITHM

B.1 EXECUTION OVERVIEW AND NOTATION

SparseEdit first performs dense warmup to initialize cached states and select an execution profile. At each subsequent step, the Execution Controller chooses among dense refresh, velocity reuse, and

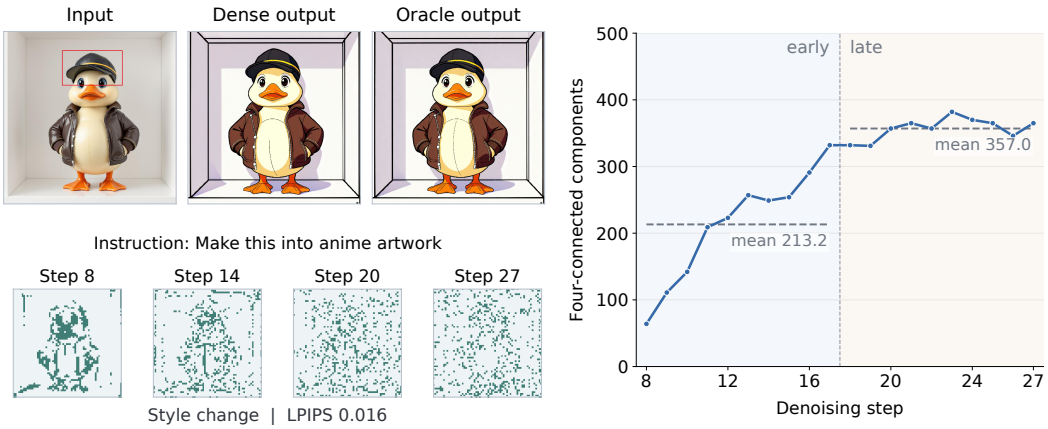


Figure 5: Style change with LongCat-Image-Edit (KontextBench request `0005_05`). With a fixed 15% budget, the LPIPS between the oracle and dense outputs is 0.016. The mean component count increases from 213.2 to 357.0.

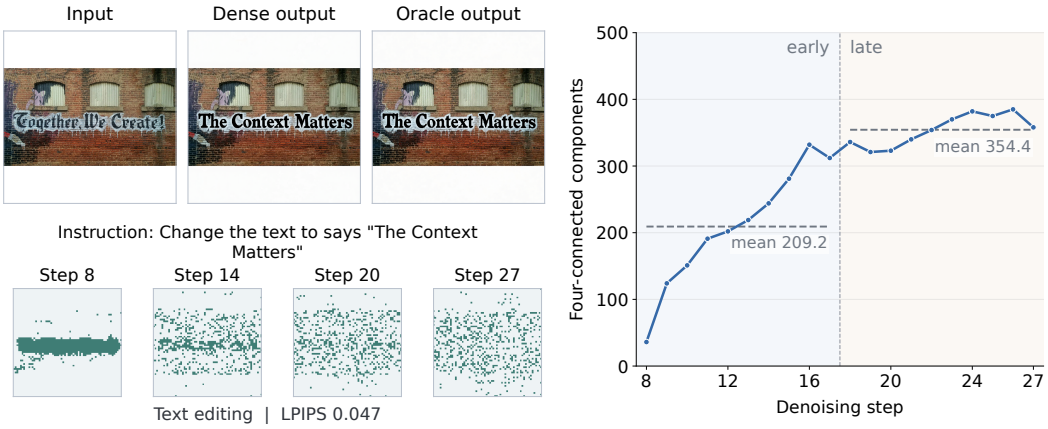


Figure 6: Text editing with LongCat-Image-Edit (KontextBench request `0014_00`). With a fixed 15% budget, the LPIPS between the oracle and dense outputs is 0.047. The mean component count increases from 209.2 to 354.4.

sparse computation. Patch selection is invoked only for sparse computation. Each action produces a scheduler velocity and updates the state required by the next step. We define these operations in execution order below, and Algorithm 1 then assembles them into the complete inference loop.

Denoising steps are indexed by $i = 0, \ldots, T-1$, producing the final latent Z_T. The normalized flow time τ_i decreases from $\tau_0 = 1$ toward $\tau_T = 0$, and $\Delta\tau_i = \tau_{i+1} - \tau_i$ denotes the scheduler step size. The latent Z_i contains N generated positions, R contains the reference-image latent tokens, and Φ denotes the DiT. Text and other auxiliary conditioning tokens are fully computed in every Transformer forward pass and are excluded from the generated-position update budget.

We use V_i for the velocity freshly computed at the current step and $\widehat{V}_i(p)$ for the most recent computed prediction cached at position p on entry to that step. The complete velocity field used by the scheduler is $\widetilde{V}_i$. The per-layer Transformer cache is $\mathcal{H}_i$. Token age $a_i(p)$ counts steps since the last computation at p, and $\mathcal{B}_i$ is the remaining velocity-reuse budget. Warmup length W and refresh period P determine the computation schedule, while routing supplies the generated-position update budget K and initial reuse budget B. A binary mask M_i identifies selected positions, and $\odot$ denotes elementwise multiplication with the mask broadcast over channels.

B.2 DENSE WARMUP AND POLICY ROUTING

Warmup runs DENSEFORWARD for $i = 0, \ldots, W-1$. At each step, this operation evaluates Φ at all generated positions, updates the per-layer cache, and returns a fresh velocity field V_i. Each warmup step uses $\widetilde{V}_i = V_i$ to advance the latent and retains V_i in the prediction cache. The resulting trajectory provides the statistics used to choose (K, B) once per request.

Distances measured during warmup. At step i, the velocity prediction gives the clean-latent estimate $\widehat{Z}_i^{\text{clean}} = Z_i - \tau_i V_i$. The mean per-token distance between consecutive clean-latent predictions over transitions $i = 1, \ldots, W-1$ is

$$\bar{d}_i^{\text{clean}} = \frac{1}{N} \sum_{p=1}^{N} \left\| \big(Z_i - \tau_i V_i\big)(p) - \big(Z_{i-1} - \tau_{i-1} V_{i-1}\big)(p) \right\|_2 . \tag{9}$$

For edit extent, let $R(p)$ denote the reference-image latent aligned with generated position p. The distance between the final warmup clean prediction and the reference is

$$d_{\text{ref}}(p) = \left\| \widehat{Z}_{W-1}^{\text{clean}}(p) - R(p) \right\|_2 . \tag{10}$$

These distances respectively describe trajectory change and departure from the source. Here latent-space distance provides a low-cost proxy for edit extent.

Warmup metrics. Using these distances, WARMUPSTATISTICS computes

$$\begin{aligned} m_{\text{late}} &= \frac{\bar{d}_{W-2}^{\text{clean}} + \bar{d}_{W-1}^{\text{clean}}}{2}, \\ m_{\text{conv}} &= \frac{m_{\text{late}}}{(\bar{d}_1^{\text{clean}} + \bar{d}_2^{\text{clean}})/2 + \epsilon}, \\ m_{\text{edit}} &= \frac{1}{N} \sum_{p=1}^{N} \mathbf{1}[d_{\text{ref}}(p) > \eta_{\text{edit}}] . \end{aligned} \tag{11}$$

Here ϵ guards the ratio, m_{late} measures late prediction change, m_{conv} compares late change with early change, and m_{edit} estimates the fraction of positions departing from the reference. With $W = 8$, the early mean uses transitions $1, 2$ and the late mean uses transitions $6, 7$. The edit-discrepancy threshold η_{edit} is shared across requests, so the coverage metric measures thresholded latent discrepancy rather than a ground-truth semantic edit region.

Routing score and execution budgets. We scale the absolute late-change magnitude by a positive reference value λ in the same latent-distance units, and the convergence ratio and edit fraction are already dimensionless. WARMUPSTATISTICS maps the three metrics to bounded scores and combines them with equal weights,

$$s_{\text{route}} = \frac{1}{3} \left(\frac{\lambda}{\lambda + m_{\text{late}}} + \frac{1}{1 + m_{\text{conv}}} + \big(1 - m_{\text{edit}}\big) \right). \tag{12}$$

Each term lies in $[0, 1]$ and decreases as its corresponding metric increases. The first preserves differences in late-change magnitude under a fixed reference across requests. The second describes temporal convergence. The third retains the complement of the estimated edit coverage. Thus λ sets the magnitude scale, while the weights specify the contribution of each transformed term.

Larger scores imply stable trajectories and smaller estimated edit regions. POLICYROUTING selects the aggressive profile when $s_{\text{route}} > \theta_{\text{route}}$ and the conservative profile otherwise. Each profile specifies an update ratio ρ and an initial reuse budget B, with a smaller ratio and larger budget in the aggressive profile. The generated-position budget is

$$K = \max\big(1, \lfloor \rho N \rfloor\big). \tag{13}$$

The resulting (K, B) governs all subsequent steps, with $\mathcal{B}_W = B$ after warmup.

B.3 Action Selection and Velocity Reuse

After warmup, the controller determines whether the current step requires a forward pass. A scheduled dense refresh, with period P, takes precedence over the reuse decision and invokes DenseForward. At other steps, a reuse score is compared with the remaining budget $\mathcal{B}_i$. We first define the velocity scaling and score used in this decision.

Velocity scaling. Prior work has extensively studied residual-based caching for diffusion models (Chen et al., 2024; Liu et al., 2024; Zhou et al., 2025; Bu et al., 2025; Chen et al., 2025). Following this velocity-decay approach, we rescale the preceding scheduler velocity using the factor

$$r_i = \big(1 + \tau_i - \tau_{i-1}\big). \tag{14}$$

If reuse is accepted, VelocityReuse forms $\widetilde{V}_i = r_i \widetilde{V}_{i-1}$ without a Transformer forward pass.

Reuse score from available history. When step $i-1$ performed a computation, ReuseScore compares consecutive scheduler velocities, scales their difference by the current scheduler step size, and normalizes it by the current latent magnitude,

$$g_i = \frac{|\tau_{i+1} - \tau_i|}{\|Z_i\|_F + \epsilon} \left\| \widetilde{V}_{i-1} - r_{i-1} \widetilde{V}_{i-2} \right\|_F . \tag{15}$$

The squared residual norm is the sum $\sum_{p=1}^{N} \|\widetilde{V}_{i-1}(p) - r_{i-1}\widetilde{V}_{i-2}(p)\|_2^2$ over generated positions. The factor $|\Delta\tau_i|$ weights this residual by the current scheduler step size. Smaller scores favor reuse and consume less budget when reuse is accepted. After sparse computation, the scheduler velocity contains both fresh and cached predictions.

When step $i-1$ itself reused velocity, the residual above vanishes because $\widetilde{V}_{i-1} = r_{i-1}\widetilde{V}_{i-2}$. The score instead extrapolates the preceding value,

$$g_i = \alpha \, g_{i-1}, \tag{16}$$

where penalty α accounts for the added uncertainty of consecutive reuse. The controller retains the preceding action and score to choose between these two rules.

Action decision. ExecutionController returns $c_i =$ Refresh at a scheduled refresh, $c_i =$ VelocityReuse when no refresh is scheduled and $g_i \le \mathcal{B}_i$, and $c_i =$ SparseCompute otherwise. The final denoising step follows the same rule. The scoring rule and α remain fixed across requests, while routing assigns only the initial budget B.

Reuse budget updates. The routed value B initializes $\mathcal{B}_W$ after warmup, and $\mathcal{B}_i$ is the balance available at the start of step i. Each accepted VelocityReuse consumes the current score g_i, giving $\mathcal{B}_{i+1} = \mathcal{B}_i - g_i$. Either Refresh or SparseCompute forward pass restores $\mathcal{B}_{i+1} = B$.

B.4 Stage-Aware Patch Selection and Sparse Computation

When neither a scheduled refresh nor velocity reuse is selected, SparseEdit constructs an update mask and performs sparse computation at the selected positions. The selector takes the two preceding scheduler velocities, token ages, current flow time, and the routed budget K as inputs. Its output M_i determines which generated positions receive fresh computation.

Base score and normalization. As in Section 4.2, the velocity change $d_i(p) = \|\widetilde{V}_{i-1}(p) - \widetilde{V}_{i-2}(p)\|_2$ is available before the current forward pass. Let Ω be the generated-token spatial grid. We multiply d_i by the token-age factor $1 + 0.5a_i$ and normalize the product across Ω:

$$s_i(p) = \mathrm{Norm}_\Omega(d_i(1 + 0.5a_i))\,(p), \tag{17}$$

For an input map x, Norm_Ω is the following min–max normalization over the N generated positions $q \in \Omega$:

$$\mathrm{Norm}_\Omega(x)(p) = \begin{cases} \dfrac{x(p) - \min_{q\in\Omega} x(q)}{\max_{q\in\Omega} x(q) - \min_{q\in\Omega} x(q)}, & \max_{q\in\Omega} x(q) - \min_{q\in\Omega} x(q) > 10^{-8}, \\ 0, & \text{otherwise}, \end{cases}$$

If the input scores differ by at most 10^{-8} across positions in some extreme situations, normalization sets all scores to zero. The age factor raises the priority of positions left uncomputed for several steps and equals one for freshly computed positions. Unlike the offline oracle, which uses adjacent full forward predictions, this online signal uses the retained scheduler velocities.

Neighborhood averaging. To encourage locally grouped updates, we average the base score over the centered 5×5 window at p, restricted to the generated-token grid,

$$\mathrm{Avg}_{5\times5}(s_i)(p) = \frac{1}{|\mathcal{N}_5(p)|} \sum_{q\in\mathcal{N}_5(p)} s_i(q), \qquad \mathcal{N}_5(p) = \big\{q \in \Omega : \|q-p\|_\infty \le 2\big\}. \tag{18}$$

Near a boundary, fewer than 25 neighbors lie inside the grid. Averaging only those neighbors keeps boundary and interior windows on the same scale.

Stage-aware interpolation and mask construction. The current flow time controls the transition from neighborhood-averaged to pointwise priorities. With τ_W denoting the flow time at the end of warmup, PATCHSELECTOR constructs

$$\begin{aligned} \bar{s}_i(p) &= \frac{\tau_W - \tau_i}{\tau_W}\, s_i(p) + \frac{\tau_i}{\tau_W}\, \mathrm{Avg}_{5\times5}(s_i)(p), \\ M_i &= \mathrm{TopKMask}(\bar{s}_i, K), \qquad \|M_i\|_0 = K. \end{aligned} \tag{19}$$

The interpolation uses averaged scores at $\tau_i = \tau_W$ and approaches the base scores as $\tau_i \to 0$, permitting progressively more dispersed updates. It preserves constant maps and the score range. Flow time advances through refresh and reuse steps as well as sparse computation, so this transition follows denoising progress even when the selector is not invoked at every step.

TOPKMASK selects K positions under the budget defined in Eq. 13, resolves ties by a fixed grid-index order, and sets $M_i(p) = 1$ at those positions. Spatially connected selected positions form update patches. The neighborhood and interpolation rule are shared across requests and profiles.

Sparse attention and cache writes. The resulting mask is passed to SPARSECOMPUTE. Let $\mathcal{A}_i = \{p : M_i(p) = 1\}$ denote the active generated positions. Their queries attend to the full cached context, which consists of fresh projections at active positions, cached projections at inactive positions, and cached reference-image latent tokens. At block ℓ, with attention head dimension d_h,

$$\mathbf{O}^{(\ell)}_{i,\mathcal{A}_i} = \mathrm{Softmax}\left(\frac{\mathbf{Q}^{(\ell)}_{i,\mathcal{A}_i}(\mathbf{K}^{(\ell)}_i)^\top}{\sqrt{d_h}}\right)\mathbf{V}^{(\ell)}_i, \tag{20}$$

Here $\mathbf{Q}^{(\ell)}_{i,\mathcal{A}_i}$ contains active queries, and $\mathbf{K}^{(\ell)}_i$ and $\mathbf{V}^{(\ell)}_i$ contain keys and values across generated-latent and reference-image latent positions. Auxiliary conditioning tokens are implicit as in Section 4.3. The attention key/value matrices are distinct from the velocity prediction V_i. Active attention outputs pass through the output projection, residual connection, and feed-forward network. New active states replace their per-layer cache entries, while inactive entries remain unchanged. SPARSECOMPUTE returns fresh velocity predictions V_i at the selected positions and the updated per-layer cache. Values outside the mask are supplied by the prediction cache when forming the complete scheduler velocity below.

B.5 STATE UPDATES AND DENOISING STEP

The three actions produce a complete scheduler velocity by different means,

$$\widetilde{V}_i = \begin{cases} V_i, & c_i = \textsc{Refresh}, \\ r_i \widetilde{V}_{i-1}, & c_i = \textsc{VelocityReuse}, \\ M_i \odot V_i + (1-M_i)\odot \widehat{V}_i, & c_i = \textsc{SparseCompute}. \end{cases} \tag{21}$$

Table 3 gives the corresponding prediction-cache, age, and budget updates. Dense computation updates all generated-position entries in $\mathcal{H}_i$, sparse computation updates only active entries as described above, and velocity reuse preserves the per-layer cache. In particular, velocity reuse changes

Table 3: State carried into step $i+1$ after each post-warmup action. Age arithmetic is positionwise.

Action c_i	Prediction cache $\widehat{V}_{i+1}$	Token age a_{i+1}	Budget $\mathcal{B}_{i+1}$
REFRESH	V_i	$\mathbf{0}$	B
VELOCITYREUSE	$\widehat{V}_i$	$a_i + \mathbf{1}$	$\mathcal{B}_i - g_i$
SPARSECOMPUTE	$\widetilde{V}_i$	$(1 - M_i) \odot (a_i + \mathbf{1})$	B

$\widetilde{V}_i$ without overwriting $\widehat{V}_i$, which continues to hold the most recent computed prediction at each position.

All branches then invoke the same STEP operation,

$$Z_{i+1} = Z_i + (\tau_{i+1} - \tau_i)\widetilde{V}_i. \tag{22}$$

The resulting latent, velocity history, and updated states are the inputs to the next denoising step. During warmup, the velocity, cache, and age updates follow the dense-computation case, and routing initializes the reuse budget only after warmup ends.

B.6 COMPLETE INFERENCE ALGORITHM

Algorithm 1 assembles the operations defined above. Its phases correspond to warmup and routing (Appendix B.2), action selection (Appendix B.3), sparse computation (Appendix B.4), and state advancement (Appendix B.5). The mutable variables $\mathcal{H}$, $\widehat{V}$, a, and $\mathcal{B}$ hold the current states, and $\widetilde{V}_{i-2:i-1}$ denotes the two preceding scheduler velocity fields. In the algorithm, B_{init} denotes the fixed routed budget B, while $\mathcal{B}$ is the remaining balance. All procedures have access to the fixed DiT, conditioning inputs, sampling schedule, and settings in Appendix C. The controller retains the preceding action and score across iterations. REUSESCORE reads that action and, after a reuse step, its score.

Algorithm 1 SparseEdit inference with warmup routing and adaptive execution

Require: Initial latent Z_0, reference tokens R, schedule $\{\tau_i\}_{i=0}^{T}$, DiT $\Phi(\cdot)$, warmup length W, refresh period P, reuse penalty α.
Ensure: Edited latent Z_T.

$\mathcal{H} \leftarrow \varnothing$; $\widehat{V} \leftarrow \varnothing$; $a \leftarrow \mathbf{0}$; $\mathcal{B} \leftarrow 0$.
// Dense warmup and policy routing (Appendix B.2)
for $i = 0, \ldots, W-1$ **do**
  $(V_i, \mathcal{H}) \leftarrow \text{DENSEFORWARD}(Z_i, R, \mathcal{H})$; $c_i \leftarrow \text{REFRESH}$.
  $\widetilde{V}_i \leftarrow V_i$; $\widehat{V} \leftarrow V_i$; $Z_{i+1} \leftarrow \text{STEP}(Z_i, \widetilde{V}_i, \tau_i, \tau_{i+1})$.
end for
$s_{\text{route}} \leftarrow \text{WARMUPSTATISTICS}(\{Z_i, V_i\}_{i=0}^{W-1}, R)$.
$(K, B_{\text{init}}) \leftarrow \text{POLICYROUTING}(s_{\text{route}}; \theta_{\text{route}})$; $\mathcal{B} \leftarrow B_{\text{init}}$.
// Select an action, then update states and latent during routed generation
for $i = W, \ldots, T-1$ **do**
  $g_i \leftarrow \text{REUSESCORE}(i, \widetilde{V}_{i-2:i-1}, Z_i; \alpha)$.
  $r_i \leftarrow (1 + \tau_i - \tau_{i-1})$; $c_i \leftarrow \text{EXECUTIONCONTROLLER}(i, g_i, \mathcal{B}; P)$.
  if $c_i = \text{REFRESH}$ **then**
    $(V_i, \mathcal{H}) \leftarrow \text{DENSEFORWARD}(Z_i, R, \mathcal{H})$; $\widetilde{V}_i \leftarrow V_i$; $\widehat{V} \leftarrow V_i$.
    $\mathcal{B} \leftarrow B_{\text{init}}$; $a \leftarrow \mathbf{0}$.
  else if $c_i = \text{VELOCITYREUSE}$ **then**
    $\widetilde{V}_i \leftarrow \text{VELOCITYREUSE}(\widehat{V}_{i-1}, r_i)$.
    $\mathcal{B} \leftarrow \mathcal{B} - g_i$; $a \leftarrow a + \mathbf{1}$.
  else
    $M_i \leftarrow \text{PATCHSELECTOR}(\widetilde{V}_{i-2:i-1}, a; K, \tau_i, \tau_W)$.
    $(V_i, \mathcal{H}) \leftarrow \text{SPARSECOMPUTE}(Z_i, R, \mathcal{H}; M_i)$.
    $\widetilde{V}_i \leftarrow M_i \odot V_i + (1 - M_i) \odot \widehat{V}$; $\widehat{V} \leftarrow \widetilde{V}_i$.
    $\mathcal{B} \leftarrow B_{\text{init}}$; $a \leftarrow (1 - M_i) \odot (a + \mathbf{1})$.
  end if
  $Z_{i+1} \leftarrow \text{STEP}(Z_i, \widetilde{V}_i, \tau_i, \tau_{i+1})$.
end for
return Z_T.

C Implementation Settings and Hyperparameters

C.1 Shared Settings and Execution Profiles

Table 4 collects the numerical settings and selection rules used in the main experiments. The refresh schedule, reference-image cache policy, scoring operators, and reuse penalty are shared across requests and profiles. Each profile supplies only its update ratio and initial reuse budget. All settings are fixed before evaluation rather than re-estimated per test example.

Table 4: SparseEdit hyperparameters and selection rules.

Parameter	Value	Selection or use
Denoising steps	$T = 28$	Shared by all three editors in the main experiments
Dense warmup	$W = 8$ steps	Shared main-experiment warmup
Dense refresh interval	$P = 15$ steps	Fixed schedule, included in measured latency
Local averaging window	5×5	In-grid mean in Eq. 18
Reuse extrapolation factor	$\alpha = 1.5$	Applied when step $i - 1$ reused velocity
Consecutive reuse	$g_i \leq \mathcal{B}_i$	Determined by remaining budget and extrapolated score
Aggressive profile	$(\rho, B) = (0.20, 0.02)$	Used when score $> \theta_{\text{route}}$
Conservative profile	$(\rho, B) = (0.40, 0.01)$	Used when score $\leq \theta_{\text{route}}$
Edit-discrepancy threshold	$\eta_{\text{edit}} = 3.0$	Threshold for distance between source and prediction latents
Late-change reference scale	$\lambda = 1$	Fixed latent-distance scale in Eq. 12
Routing weights	$1/3$	Equal weights over the three transformed terms
Routing threshold	$\theta_{\text{route}} = 0.50$	Single profile boundary

D Mask Connectivity Under SparseEdit Selection

The oracle analysis in Appendix A uses full velocity fields. To assess whether its early-to-late increase in mask fragmentation also appears under an inference-time signal, we examine masks selected before the current DiT forward pass. Figure 7 reports four-connected component counts for masks produced with the selectivity prior on the 64×64 generated-token grid for one background-change example.

This diagnostic performs as an ablation study for the main experiment in Section 5.3. Velocity reuse is disabled to show more metrics. Both configurations use $T = 28$ denoising steps and a refresh interval of 15 steps.

As shown in figure 7, the selected mask has 6 components at step 8 and 319 at step 27. The mean component count rises from 94.7 over steps 8–17 to 208.6 over steps 18–27, excluding the refresh at step 23. From the curve, we can see that SparseEdit simulates the mask variation law of oracle analysis.

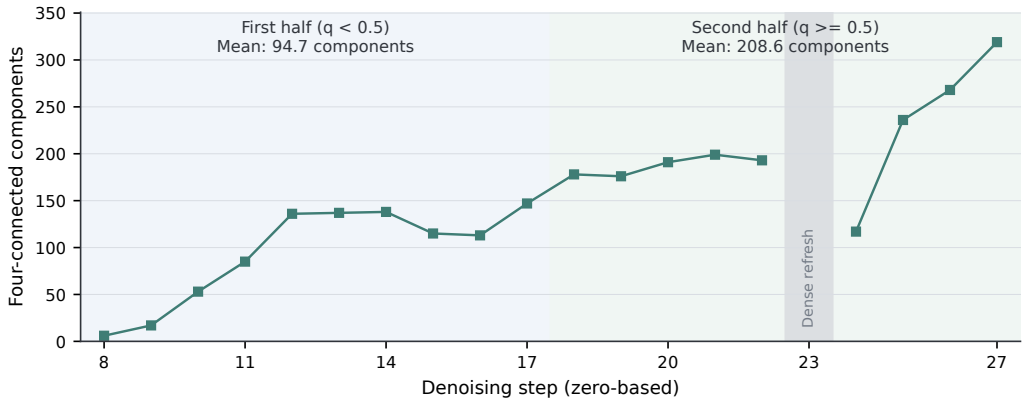


Figure 7: Connectivity of Patch Selector masks for a GEdit-Bench background-change case with FLUX.1 Kontext Dev. The dense refresh at step 23 is shown as a gap.