HASTE: HIERARCHICAL ADAPTIVE SPATIO-TEMPORAL REUSE FOR EFFICIENT IMAGE DECOMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Text-to-layer (T2L) diffusion models achieve editable image decomposition by jointly denoising multi-layer RGBA stacks, but suffer from prohibitive inference latency. Existing diffusion accelerators overlook this multi-layer structure, treating images as flat, spatially contiguous domains. We observe that computational redundancy in T2L models is inherently hierarchical, governed by **inter-layer temporal heterogeneity** (layers stabilize at disparate denoising stages) and **intra-layer spatial sparsity** (active synthesis is strictly confined to a sparse **alpha support**, outside of which latent states quickly freeze). To exploit this insight, we propose **HASTE**, the first dedicated online caching framework for T2L generation. **HASTE** infers alpha support online via an ultra-lightweight token-level regressor and deploys a two-stage hierarchical scheduler: converged layers bypass Transformer execution entirely via flow-matching endpoint extrapolation, while active layers restrict fresh evaluation strictly to tokens within the alpha support. To prevent cross-layer context fragmentation, **HASTE** executes a sparse DiT scheme powered by persistent Key-Value caches. On the Crello benchmark, **HASTE** accelerates Qwen-Image-Layered sampling by $3.22\times$ with negligible quality loss. Furthermore, it generalizes training-free across four diverse T2L backbones, delivering consistent $1.52\times$–$4.26\times$ speedups. Code is available at `https://anonymous.4open.science/r/HASTE-0434`.

1 INTRODUCTION

Text-to-layer (T2L) generation has emerged as a compelling paradigm for structured visual synthesis, decomposing an RGB image into an ordered stack of editable RGBA layers (Suzuki et al., 2025; Liu et al., 2025d; Tudosiu et al., 2024; Yang et al., 2025). By decoupling foreground elements, decorative assets, and background contexts, layered representations enable fine-grained downstream editing and graphic design workflows while preserving overall scene harmony. Modern state-of-the-art T2L architectures, such as Qwen-Image-Layered (Yin et al., 2025), formulate this task as a joint flow-matching process (Lipman et al., 2023; Esser et al., 2024) where a unified Diffusion Transformer (DiT) (Peebles & Xie, 2023) simultaneously denoises all layers. While joint modeling captures inter-layer spatial composition and mutual occlusions (Huang et al., 2024; 2025), it incurs prohibitive inference latency: the total sequence length scales directly with the number of layers. As a result, standard sampling repeatedly executes dense multi-head self-attention (Vaswani et al., 2017; Dao et al., 2022; Dao, 2024) and feed-forward networks across all layers and spatial patches at every step, regardless of their actual rate of change.

To reduce inference costs, caching methods reuse intermediate representations across timesteps (Ma et al., 2024b; Zhao et al., 2025; Chen et al., 2024b; Selvaraju et al., 2024; Lv et al., 2025), dynamically guided by runtime heuristics (Liu et al., 2025a; Bu et al., 2026) or token sensitivity (Zou et al., 2025; Chen et al., 2026; Lou et al., 2024). However, existing techniques inherently assume single-layer, spatially contiguous images. **To our knowledge, no dedicated caching framework exists for T2L generation.** Directly applying generic accelerators to multi-layer generation induces severe sub-optimality due to two domain discrepancies: (1) *Transparent canvas disturbance*: Since non-background layers are mostly transparent, latent noise fluctuations in empty regions are mistaken for active synthesis, wasting compute on void space; (2) *Layer-scale disparity*: Extreme scale variations across RGBA layers bias global spatial averaging—diluting update signals in compact foregrounds

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.

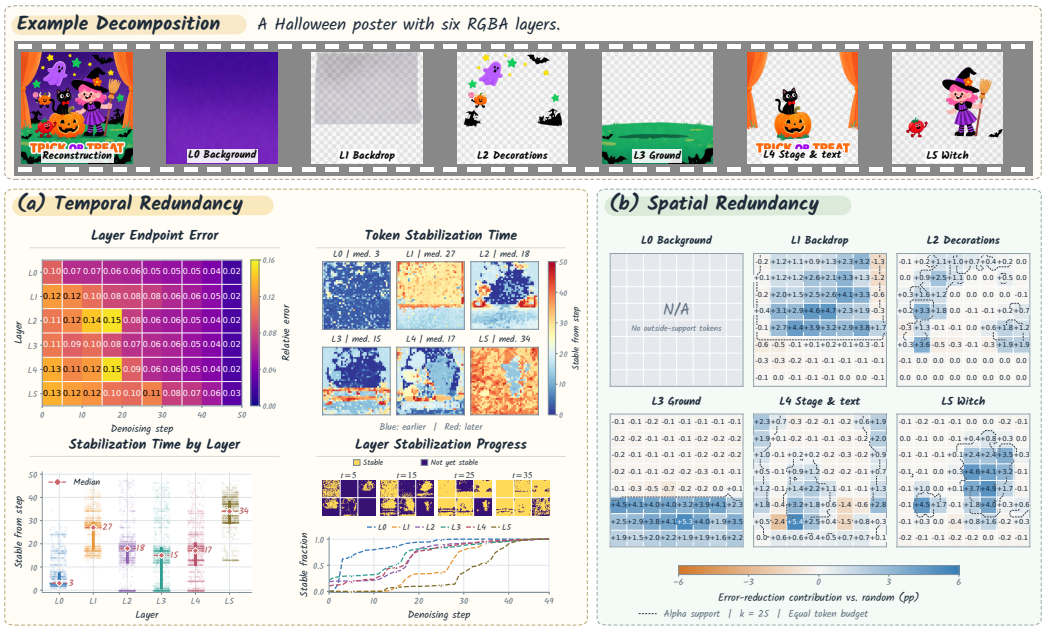

Figure 1: **Two-level redundancy in T2L generation. (a) Temporal:** output layers stabilize at different denoising stages. **(b) Spatial:** Endpoint reuse outside the alpha support yields lower error than equal-count random reuse. Cells show contributions to whole-layer error reduction (percentage points).

(triggering premature freezing) while over-computing already converged large layers. Hence, the key question is not simply *whether* to reuse computation, but *when* and *where* fresh computation is genuinely needed for T2L generation.

By systematically examining full-computation T2L trajectories, we observe that computational redundancy is inherently *hierarchical*, characterized by **inter-layer temporal heterogeneity** and **intra-layer spatial sparsity**. Across layers, different semantic components stabilize at starkly different denoising stages: coarse background layers often solidify early, whereas intricate foreground objects require prolonged refinement. Within each layer, semantic content is confined to a sparse region of non-zero opacity, which we denote as the **alpha support** ($\mathcal{S}_{t,i}$). Regions outside this support ($\alpha \approx 0$) stabilize almost immediately. Crucially, this alpha support acts as an authoritative, domain-native geometric prior: *it defines both the optimal spatial domain for monitoring layer convergence and the precise active token subset requiring fresh evaluation.*

Translating this redundancy into wall-clock speedup, however, poses three non-trivial challenges: (i) Neither the final alpha maps nor the corresponding alpha supports are accessible during sampling. (ii) The endpoint change of a skipped layer cannot be observed without evaluating the model. (iii) Removing skipped tokens from the DiT would discard the cross-layer context required for joint generation. Thus, an effective accelerator should determine which layers and tokens to refresh from inexpensive online signals, while preserving the global context needed by sparse computation.

To address these challenges, we present HASTE. First, to overcome support inaccessibility, HASTE introduces *AlphaNet*, an ultra-lightweight regressor that predicts token opacity from cached latents to derive a dilated alpha support mask $\mathcal{S}_{t,i}$. Second, HASTE adopts a **two-stage hierarchical scheduler**: (i) a *temporal stage* that skips converged layers via endpoint extrapolation, guided by feature updates pooled strictly over the support and normalized by layer sensitivity; and (ii) a *spatial stage* that restricts fresh evaluation in active layers strictly to tokens in $\mathcal{S}_{t,i}$. Third, to avert context fragmentation, we devise a context-preserving sparse DiT execution mechanism with persistent KV caching: queries and feed-forward networks run only on active tokens, while keys and values incorporate cached representations to preserve full cross-layer interaction. Periodic global refreshes and terminal refinement further suppress cumulative drift.

Our main contributions are summarized as follows:

- We identify a hierarchical redundancy structure fundamental to T2L diffusion, revealing that inter-layer convergence rates vary markedly and that intra-layer updates are spatially governed by the layer's alpha support.
- We propose **HASTE**, the first dedicated online caching framework for T2L generation, coupling lightweight alpha-support estimation with a two-stage scheduler, flow-matching endpoint extrapolation, and context-preserving sparse DiT execution.
- Extensive evaluations on the Crello benchmark demonstrate that **HASTE** accelerates Qwen-Image-Layered sampling by up to $3.48\times$ (averaging $3.22\times$) with negligible degradation in layer decomposition quality. Furthermore, **HASTE** seamlessly transfers to four diverse T2L architectures, delivering consistent $1.52\times$–$4.26\times$ speedups without any retraining of the underlying diffusion backbones.

2 Motivation

T2L generation jointly denoises all RGBA layers under one shared schedule, yet the layers evolve at different rates and are often spatially sparse.

Temporal redundancy. Fig. 1(a) shows that different output layers follow markedly different convergence trajectories. Some layers become largely stable early in denoising, while others continue to evolve much later. Treating all layers identically therefore wastes computation on layers whose predictions have already changed little. Instead, the update frequency should adapt to the state of each layer.

Spatial redundancy. An RGBA layer is represented over the full canvas, but its content is typically concentrated within a sparse alpha support, leaving many positions nearly transparent. Fig. 1(b) shows that reusing predictions outside this support incurs less error than reusing the same number of randomly selected positions within the layer. This makes alpha support a natural cue for deciding where the computation should actually be spent.

3 Method

Motivated by the temporal and spatial redundancies observed in Sec. 2, we propose **HASTE**, an alpha-guided hierarchical caching framework for accelerating T2L generation. At each denoising step, **HASTE** executes a dual-level caching policy: stabilized layers are advanced directly via cached clean endpoints, while active layers compute fresh attention exclusively for unconverged tokens, recycling Key-Value (KV) representations elsewhere. Below, we formalize the baseline T2L formulation before detailing our hierarchical scheduling, sparse execution, and refinement pipeline.

3.1 Preliminaries: Flow-Matching-Based T2L Generation

Following recent image decomposition formulations (Yin et al., 2025), T2L generation maps an RGB image $I \in \mathbb{R}^{H\times W\times 3}$ to N ordered RGBA layers $L = \{L_i\}_{i=1}^{N} \in \mathbb{R}^{N\times H\times W\times 4}$ conditioned on text embeddings $e_{\text{txt}} \in \mathbb{R}^{n_{\text{txt}}\times d_{\text{txt}}}$. Each layer $L_i = [\text{RGB}_i; \alpha_i]$ comprises an RGB canvas and an alpha matte $\alpha_i \in [0,1]^{H\times W}$ ($\alpha = 0$ transparent, $\alpha = 1$ opaque) (Porter & Duff, 1984; Levin et al., 2008). A pre-trained RGBA-VAE (Kingma & Welling, 2014; Rombach et al., 2022) encoder $\mathcal{E}$ independently encodes each layer into clean latents $x_0 = \mathcal{E}(L) \in \mathbb{R}^{N\times h\times w\times c}$, and encodes the input image as condition $z_I = \mathcal{E}([I; \mathbf{1}]) \in \mathbb{R}^{h\times w\times c}$.

Under flow matching (Liu et al., 2023; Albergo & Vanden-Eijnden, 2023), at step t with noise level $\sigma_t \in [0,1]$, noisy latents are denoted by $z_t \in \mathbb{R}^{N\times h\times w\times c}$. Given the model-predicted velocity v_t, the clean endpoint $\widehat{x}_t$ is predicted and the latent advances to z_{t+1} via:

$$\widehat{x}_t = z_t - \sigma_t v_t, \qquad z_{t+1} = z_t + (\sigma_{t+1} - \sigma_t) v_t. \tag{1}$$

The backbone DiT (Chen et al., 2024a; Li et al., 2024b) processes flattened token sequences produced by a spatial patchification operator (Dosovitskiy et al., 2021) $\mathcal{P} : \mathbb{R}^{h\times w\times c} \to \mathbb{R}^{S\times d_z}$, where $S = (h/p_h)(w/p_w)$ is the number of spatial tokens per layer for a patch size $p_h \times p_w$, and $d_z = p_h p_w c$ denotes the flattened token dimension.

Notations. We let $i \in \{1,\dots,N\}$ index an editable layer, and $p \in \{1,\dots,S\}$ index a spatial patch token. The noisy latent, velocity, and predicted endpoint of patch p in layer i are denoted by

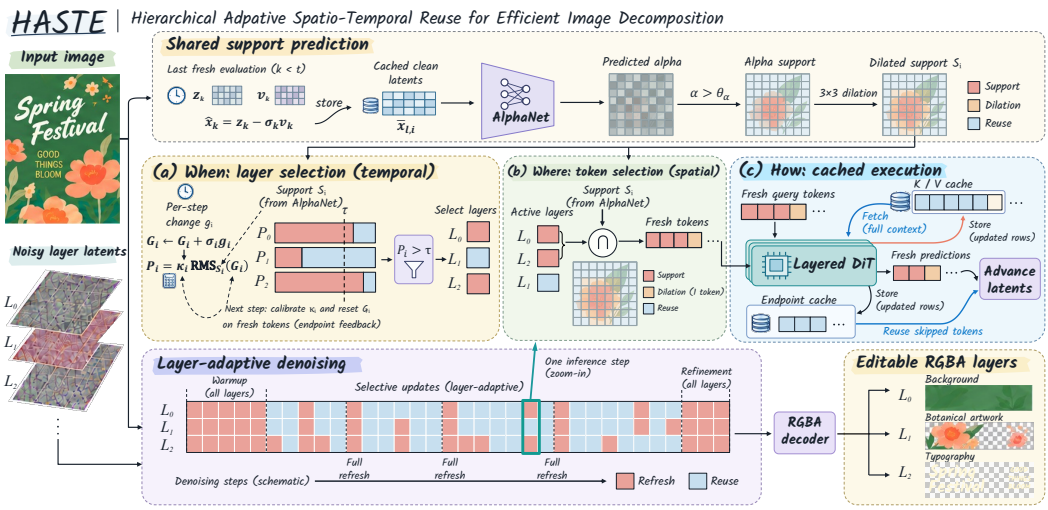

Figure 2: **Overview of HASTE.** At each denoising step, the Alpha-Guided Hierarchical Scheduler predicts token-level alpha support, dynamically selecting active layers and unconverged tokens within them. Active tokens are updated via context-preserving sparse DiT execution with persistent KV caches, while skipped layers and background tokens advance along trajectories extrapolated from cached endpoints.

$z_{t,i,p}, v_{t,i,p}, \widehat{x}_{t,i,p} \in \mathbb{R}^{d_z}$, respectively. The cached endpoint from its latest evaluation is denoted by $\overline{x}_{t,i,p} \in \mathbb{R}^{d_z}$, and the full-layer cached latent is $\overline{x}_{t,i} \in \mathbb{R}^{h \times w \times c}$. Across the N editable layers full-canvas frame, there are $M = NS$ total generated image tokens.

3.2 Alpha-Guided Hierarchical Scheduling

Building upon the empirical findings in Sec. 2, our objective is to eliminate computational waste through joint layer-level and token-level caching. Specifically, for layers whose latent states have stabilized, full Transformer evaluation can be completely bypassed by reusing cached endpoints. For layers that remain in active synthesis, token-level computation should be concentrated exclusively on regions that have not yet converged, while recycling the attention Key-Value (KV) states of stabilized regions.

The key challenge lies in *online attribution*: **How can the model identify stabilized layers and converged tokens prior to committing to heavy Transformer computation?** Existing text-to-image/video acceleration frameworks typically monitor representation or velocity differences across steps to gauge convergence (Kahatapitiya et al., 2025; Zhou et al., 2025). However, this heuristic fails in multi-layer image decomposition due to two domain discrepancies: **(1) transparent canvas disturbance:** decomposed layers are predominantly transparent. In empty regions, stochastic latent noise fluctuations are easily misinterpreted as active updates, causing models to waste computation on empty space; **(2) layer-scale disparity:** severe scale imbalance across RGBA layers (e.g., expansive backgrounds vs. localized foregrounds) skews global spatial averaging—diluting updates in compact layers to trigger premature freezing, while over-computing large layers.

The Alpha-Support Insight: The transparency (α) channel provides an intrinsic geometric prior unique to T2L generation. High-cost denoising is strictly required within non-zero opacity regions, which we define as the **alpha support** (Fig. 1(b)). Crucially, transparent regions ($\alpha \approx 0$) stabilize almost immediately. The spatial alpha support thus acts as an authoritative, unified prior: *it defines both the spatial domain for evaluating layer convergence and the exact active token set requiring fresh computation.*

Online Alpha-Support Estimation. Since the true alpha matte α_i is only accessible after the final RGBA-VAE decoding, running decoding at every intermediate step is computationally prohibitive. To overcome this, we train a lightweight token-level predictor, termed **AlphaNet** ($A_\psi : \mathbb{R}^{S \times d_z} \to [0,1]^S$). App. B.1 gives the training details of AlphaNet. AlphaNet directly infers the continuous

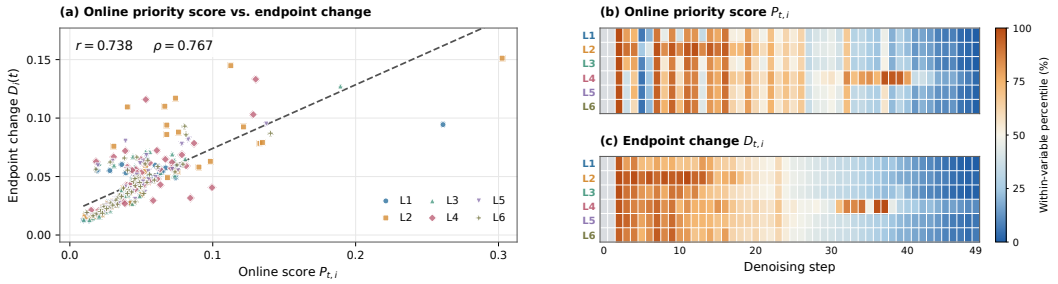

Figure 3: **Online priority scores track ground-truth endpoint changes.** Causal replay on a full-computation trajectory shows that (a) $P_{t,i}$ correlates strongly with the measured endpoint change after evaluation ($r = 0.738$, $\rho = 0.767$, $p < 0.01$), where r and ρ are the Pearson and Spearman correlations, and p is the test p-value. (b,c) Layer–step heatmaps show that the online scores closely follow the temporal variation in endpoint changes across layers.

token-grid opacity $\widehat{\alpha}_{t,i}$ from the cached latent endpoint $\overline{x}_{t,i}$, yielding the binary support set $\mathcal{S}_{t,i}$:

$$\mathcal{S}_{t,i} = \{p \in \{1, \dots, S\} \mid m_{t,i,p} = 1\}, \quad m_{t,i} = \mathrm{Dilate}_{3\times3}(\mathbb{I}[\widehat{\alpha}_{t,i} > \theta_\alpha]), \quad \widehat{\alpha}_{t,i} = A_\psi(\mathcal{P}(\overline{x}_{t,i})), \tag{2}$$

where $\theta_\alpha \in (0, 1)$ is a binarization threshold, and $\mathbb{I}[\cdot]$ is the indicator function. The 3×3 morphological dilation expands the boundary by a one-token margin on the $(h/2) \times (w/2)$ lattice, effectively hedging against support boundary jitter and small estimation errors. The resulting support set $\mathcal{S}_{t,i}$ acts as the foundational mask for our two-stage hierarchical scheduling.

Hierarchical Scheduling and Adaptive Execution. Equipped with $\mathcal{S}_{t,i}$, the scheduler executes a two-stage decision process:

1) Temporal Layer Selection and Extrapolation. Because skipped layers expose no current outputs, we use feature changes before the full Transformer as a lightweight proxy for their evolution. Prior cache-based methods show that such changes correlate with downstream feature updates (Liu et al., 2025a; Son et al., 2026; Shen et al., 2025), we therefore monitor an inexpensive feature-change proxy at the input to the first block. Let $h_{t,i,p} \in \mathbb{R}^{d_h}$ denote the time-modulated feature for token (i, p) with hidden dimension d_h. The step-wise change $g_{t,i,p}$ and unresolved accumulated change $G_{t,i,p}$ are:

$$g_{t,i,p} = \frac{\mathrm{RMS}(h_{t,i,p} - h_{t-1,i,p})}{\max(\mathrm{RMS}(h_{t-1,i,p}), \varepsilon)}, \qquad G_{t,i,p} \leftarrow G_{t-1,i,p} + \sigma_t g_{t,i,p}, \tag{3}$$

where $\varepsilon > 0$ is a numerical stabilizer. To remove background canvas noise and size bias, we pool $G_{t,i,p}$ strictly over $\mathcal{S}_{t,i}$ and scale it with a layer-specific sensitivity $\kappa_{t,i}$ to get online priority score $P_{t,i}$:

$$P_{t,i} = \kappa_{t,i} B_{t,i}, \qquad B_{t,i} = \sqrt{\frac{1}{|\mathcal{S}_{t,i}|} \sum_{p \in \mathcal{S}_{t,i}} G_{t,i,p}^2}. \tag{4}$$

A layer is selected for model evaluation if $P_{t,i} > \tau$ (with threshold τ); otherwise, it is skipped. As shown in Fig. 3, $P_{t,i}$ exhibits a strong correlation with ground-truth endpoint changes. Whenever layer i is re-evaluated, its sensitivity $\kappa_{t,i}$ is calibrated via exponential moving average (EMA):

$$\kappa_{t+1,i} = \mathrm{EMA}(\kappa_{t,i}, \widetilde{\kappa}_{t,i}), \qquad \widetilde{\kappa}_{t,i} = \frac{\mathrm{RMS}_{\mathcal{S}_{t,i}}(\widehat{x}_{t,i} - \overline{x}_{t,i})}{\max(\mathrm{RMS}_{\mathcal{S}_{t,i}}(\overline{x}_{t,i}), \varepsilon)(B_{t,i} + \varepsilon)}. \tag{5}$$

Upon evaluation, $G_{t,i,p}$ is reset to zero for updated tokens, whereas skipped tokens preserve their accumulation.

For layers identified as stable ($P_{t,i} \leq \tau$), full model evaluation is bypassed entirely. To prevent trajectory freezing, we advance the entire layer's latent $z_{t,i} \in \mathbb{R}^{h \times w \times c}$ via whole-layer endpoint extrapolation:

$$z_{t+1,i} = z_{t,i} + (\sigma_{t+1} - \sigma_t)\widetilde{v}_{t,i}, \qquad \widetilde{v}_{t,i} = \frac{z_{t,i} - \overline{x}_{t,i}}{\sigma_t}. \tag{6}$$

This allows converged layers to progress smoothly along their trajectory at zero DiT computational cost.

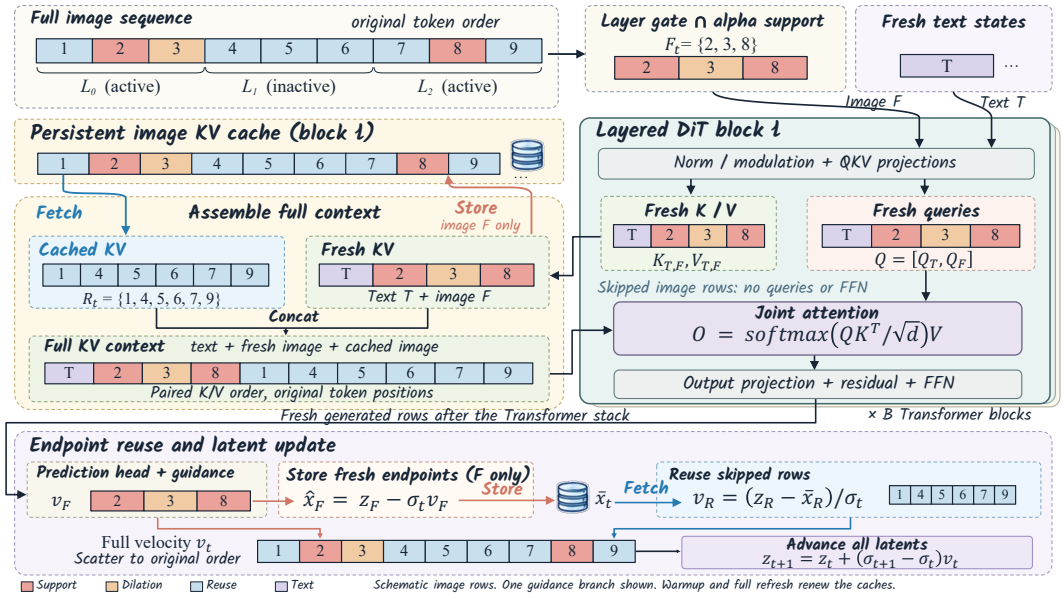

Figure 4: **Context-preserving sparse DiT execution.** Only text and active image tokens issue Queries (Q). Persistent caches provide full Key (K) and Value (V) context from skipped tokens, preserving cross-layer spatial alignment.

2) Spatial Token Selection and Context-Preserving Sparse Attention. For layers flagged as active ($P_{t,i} > \tau$), computation is further pruned at the token level using the alpha support $\mathcal{S}_{t,i}$. Specifically, transparent tokens outside the support ($p \notin \mathcal{S}_{t,i}$) are stable and cached; their corresponding latent patches are advanced via token-level endpoint extrapolation:

$$z_{t+1,i,p} = z_{t,i,p} + (\sigma_{t+1} - \sigma_t)\widetilde{v}_{t,i,p}, \qquad \widetilde{v}_{t,i,p} = \frac{z_{t,i,p} - \overline{x}_{t,i,p}}{\sigma_t} \tag{7}$$

Conversely, unconverged tokens inside the alpha support require fresh computation. These tokens form the active set:

$$\mathcal{F}_t = \{(i,p) \mid P_{t,i} > \tau,\ p \in \mathcal{S}_{t,i}\}. \tag{8}$$

Only tokens in $\mathcal{F}_t$ are fed into the DiT for fresh attention. To prevent contextual fragmentation across interdependent layers, we employ an asymmetric sparse attention mechanism backed by persistent KV caches, as shown in Fig. 4. Let $\mathcal{R} = \{1, \ldots, M\}$ index all image tokens, $\mathcal{F}_t \subseteq \mathcal{R}$ denote all active tokens ($m_t = |\mathcal{F}_t|$), and $\mathcal{R}_t = \mathcal{R} \setminus \mathcal{F}_t$ denote skipped tokens. For block ℓ with head dimension d_a, Queries are computed only for text and active tokens, while Keys and Values concatenate active states with cached states $\overline{K}^{\ell}_{\mathcal{R}_t}, \overline{V}^{\ell}_{\mathcal{R}_t}$ from skipped tokens:

$$O^{\ell}_t = \mathrm{Attn}\left([Q^{\ell}_{\mathrm{txt}}; Q^{\ell}_{\mathcal{F}_t}], [K^{\ell}_{\mathrm{txt}}; K^{\ell}_{\mathcal{F}_t}; \overline{K}^{\ell}_{\mathcal{R}_t}], [V^{\ell}_{\mathrm{txt}}; V^{\ell}_{\mathcal{F}_t}; \overline{V}^{\ell}_{\mathcal{R}_t}]\right) \in \mathbb{R}^{(n_{\mathrm{txt}}+m_t)\times d_a}. \tag{9}$$

Skipped tokens completely bypass Query generation and feed-forward networks (FFNs). After each block, newly computed KV entries update the persistent cache: $\overline{K}^{\ell}_{\mathcal{F}_t} \leftarrow K^{\ell}_{\mathcal{F}_t}, \overline{V}^{\ell}_{\mathcal{F}_t} \leftarrow V^{\ell}_{\mathcal{F}_t}$. This scales attention complexity down to $\mathcal{O}((n_{\mathrm{txt}} + m_t)(n_{\mathrm{txt}} + M))$ and FFN costs by m_t/M while rigorously preserving cross-layer spatial context.

3.3 Anchor Steps and Safety Mechanisms

To establish stable initial features, prevent cumulative drift in cached endpoints, and eliminate minor boundary inconsistencies during extended sparse sequences, **HASTE** incorporates three safety mechanisms throughout the sampling trajectory:

- **Warmup:** During the initial K_{warm} steps, all image tokens are evaluated with the full DiT to initialize endpoint and KV caches and establish reliable early scheduling states.

- **Periodic Full Refresh:** Every K_{ref} steps, a global full update is enforced across all image tokens. This renews the full KV caches, recalibrates layer sensitivities $\kappa_{t,i}$, and recovers any subtle structures emerging outside previous supports.
- **Final Refinement:** During the terminal K_{final} steps of sampling, all sparse caching is disabled in favor of standard full DiT execution. This joint refinement eliminates minor boundary inconsistencies and ensures sharp alpha transitions prior to final RGBA decoding.

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Models and Baselines. We evaluate on Crello test set (Yamaguchi, 2021) using the public 20B Qwen-Image-Layered model (Yin et al., 2025). Baselines include uncached inference (Full), PAB (Zhao et al., 2025), ToCa (Zou et al., 2025), and Jano (Chen et al., 2026). We also assess transfer to Stable-Layers (Rowles et al., 2026), CLD (Liu et al., 2025d), RevealLayer (Wang et al., 2026), and Ming-Image-0.1-Design-Layer (Ming) (inclusionAI, 2026), each against its native Full baseline.

Quality and Efficiency Metrics. We evaluate decomposition quality against ground-truth layers using LayerD's RGB L_1 and soft alpha IoU at up to five layer edits (Suzuki et al., 2025). We compare accelerated composites with uncached full inference using LPIPS (Zhang et al., 2018), SSIM (Wang et al., 2004), and PSNR. We report FLOPs, sampling latency, and speedup; latency includes online overhead but excludes model loading and final decoding.

Implementation Details. Qwen experiments use BF16, 50 denoising steps, normalized CFG 4, and seed 777 on two 48-GB RTX A6000 GPUs. Inputs retain their aspect ratios with a nominal 512-pixel short side. Comparisons share inputs, conditioning, layer counts, and initial noise. **HASTE** uses $\tau = 0.16$, $\theta_\alpha = 0.05$, and EMA weight $\lambda = 0.2$, with $K_{\text{warmup}} = 6$, $K_{\text{ref}} = 10$, and $K_{\text{final}} = 1$ for warmup, periodic refresh, and final refinement, respectively. Appendix A provides evaluation subsets, refresh schedules, cache settings, and baseline adaptations.

Table 1: Efficiency and decomposition quality on Crello test set. LayerD metrics use ground-truth layers; LPIPS, SSIM, and PSNR compare each accelerated composite with uncached full inference's.

Method	Efficiency			Visual Quality				
	FLOPs (P)↓	Speedup↑	Latency (s)↓	αIoU↑	RGB L_1 ↓	LPIPS↓	SSIM↑	PSNR↑
Qwen-Image-Layer-20B	17.261	1.00	99.03	0.8874	0.0662	–	–	–
PAB (Zhao et al., 2025)	14.317	1.22	80.85	0.8867	0.0674	0.0038	0.9911	35.36
ToCa (Zou et al., 2025)	8.275	1.99	49.62	0.8809	0.0707	0.0451	0.9516	28.77
Jano (Chen et al., 2026)	7.452	1.88	52.55	0.8864	0.0677	0.0055	0.9905	35.60
HASTE ($\tau = 0.08$)	4.871	2.75	36.06	0.8874	0.0671	0.0079	0.9877	35.56
HASTE ($\tau = 0.16$)	4.299	3.22	30.72	0.8871	0.0678	0.0081	0.9872	35.54
HASTE ($\tau = 0.20$)	4.041	3.48	28.50	0.8854	0.0676	0.0085	0.9872	34.37

4.2 EFFICIENCY AND DECOMPOSITION QUALITY

Quantitative results. Table 1 shows that **HASTE** substantially reduces both FLOPs and sampling latency while preserving decomposition quality close to uncached inference. At the default setting, **HASTE** achieves a 3.22× sampling speedup, while maintaining alpha IoU and RGB L_1. As visualized in Fig. 5, **HASTE** shifts the quality–speed operating region toward substantially higher speedups than the compared caching baselines. Increasing the admission threshold τ further improves efficiency at the cost of a modest quality change, providing a controllable speed–quality trade-off.

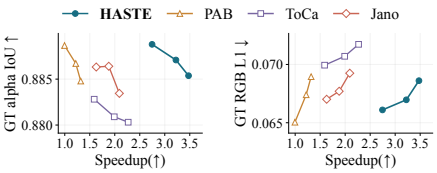

Figure 5: **Illustrative quality–speed trade-off on Qwen-Image-Layer.**

Qualitative results. Fig. 6 shows a selected example. **HASTE** preserves the background, decoration, and typography in the same layers as Full steps while accelerating sampling.

Table 2: Transfer across T2L models. Each **HASTE** result is compared with its own uncached baseline using the same sampling steps T on Crello test set.

Model	Method	Efficiency		Visual Quality				
		Speedup↑	Latency (s)↓	αIoU↑	RGB L_1 ↓	LPIPS↓	SSIM↑	PSNR↑
Stable-Layers ($T = 50$)	w/o Cache	1.00	209.90	0.8770	0.1094	–	–	–
(Rowles et al., 2026)	**HASTE**	4.26	49.22	0.8689	0.1181	0.0104	0.9869	30.60
RevealLayer ($T = 30$)	w/o Cache	1.00	220.84	0.7783	0.1415	–	–	–
(Wang et al., 2026)	**HASTE**	1.93	114.54	0.7763	0.1409	0.0320	0.9659	29.01
CLD ($T = 28$)	w/o Cache	1.00	78.30	0.8454	0.0928	–	–	–
(Liu et al., 2025d)	**HASTE**	1.70	46.00	0.8471	0.0903	0.0219	0.9750	29.59
Ming ($T = 12$)	w/o Cache	1.00	103.71	0.8829	0.0854	–	–	–
(inclusionAI, 2026)	**HASTE**	1.52	68.30	0.8812	0.0910	0.0117	0.9849	32.98

Table 3: Component ablations. T: temporal layer selection; S: spatial token selection; F: periodic full refresh during caching. Qwen uses full computation at every step. All variants retain warmup and final full updates on Crello test set.

Method				Efficiency		Visual Quality				
Component	T	S	F	Speedup↑	Latency (s)↓	αIoU↑	RGB L_1 ↓	LPIPS↓	SSIM↑	PSNR↑
Qwen-Image-Layer-20B	×	×	×	1.00	97.31	0.9110	0.0557	–	–	–
HASTE	✓	×	✓	3.07	31.74	0.9088	0.0561	0.0071	0.9879	35.70
	×	✓	✓	1.55	62.95	0.9030	0.0615	0.0114	0.9795	33.59
	✓	✓	×	3.33	29.25	0.8855	0.0692	0.1148	0.8993	25.37
	✓	✓	✓	3.19	30.49	0.9095	0.0562	0.0072	0.9878	35.67

4.3 Transfer Across T2L Models

Table 2 evaluates **HASTE** on four additional T2L models, each against its own uncached baseline. **HASTE** consistently reduces sampling time across architectures and sampling schedules, with larger gains on models that offer more timesteps for reuse. The variation in speedup also reflects differences in denoising length and the fraction of steps available for selective reuse. Decomposition quality remains close to full steps. Overall, these results demonstrate that **HASTE** transfers beyond the Qwen backbone and remains effective across diverse T2L generation pipelines.

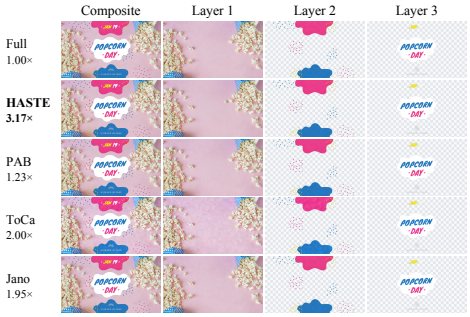

Figure 6: **Qualitative comparison of layer decomposition.**

4.4 Ablation Study

Table 3 isolates the contributions of temporal layer selection, spatial token selection, and periodic full refresh. Temporal selection accounts for most of the acceleration, while adding spatial selection further improves efficiency with little change in decomposition quality. Spatial selection alone provides less speedup and degrades quality, confirming that spatial reuse is most effective after layer-level filtering. Finally, periodic full refresh slightly reduces speedup but substantially restores both layer quality and composite fidelity.

4.5 Hyperparameter Sensitivity Analysis

Fig. 7 shows that τ is the main control knob for **HASTE**'s speed–quality trade-off: a larger threshold enables more aggressive reuse and higher speedup. In contrast, **HASTE** is relatively robust to the alpha threshold θ, EMA weight λ and periodic full refresh steps K_{ref}, which cause only minor variation over a broad range. Warmup length show the expected trade-off between reliable initialization and reuse opportunity, with a moderate number of steps providing the best balance.

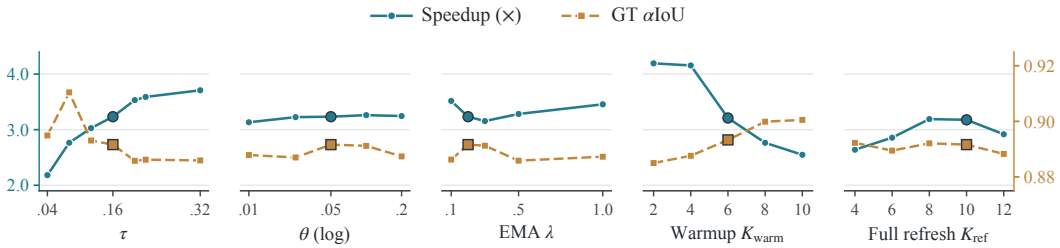

Figure 7: **Hyperparameter sensitivity.**

4.6 Runtime breakdown analysis

Figure 8 shows a consistent runtime pattern across T2L backbones. HASTE reduces most of the intermediate denoising cost through selective execution, while the remaining latency is largely attributable to full-computation warmup. Alpha support estimation and scheduling only introduce modest overhead, showing that HASTE's hierarchical policy remains lightweight across model architectures.

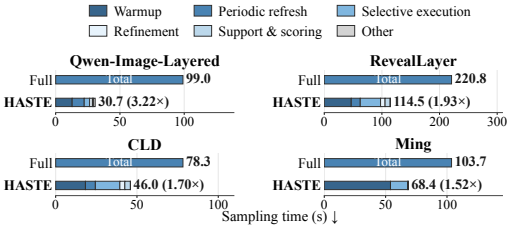

Figure 8: **Component-wise runtime breakdown across T2L models.**

5 Related Work

Editable Layer Generation and Decomposition. Layered representations support independent editing (Lu et al., 2021; Kasten et al., 2021; Zhang et al., 2023; Sarukkai et al., 2024) through transparent image generation and latent-layer manipulation (Zhang & Agrawala, 2024; Jia et al., 2025). LayerD, Qwen-Image-Layered, and CLD recover editable foreground or RGBA layers from a composited input through iterative extraction, joint generation, or conditional control (Suzuki et al., 2025; Yin et al., 2025; Liu et al., 2025d). Stable-Layers and RevealLayer further improve decomposition through VLM-scored reinforcement learning and reconstruction of visible and occluded content, respectively (Rowles et al., 2026; Wang et al., 2026). HASTE uses this RGBA structure to accelerate inference, allocating computation across output layers and within their alpha supports while keeping backbone parameters fixed.

Caching for Diffusion Acceleration. Diffusion caching exploits temporal redundancy (Zou et al., 2026) by reusing slowly varying U-Net features (Ma et al., 2024b; Li et al., 2024a; Wimbauer et al., 2024; Liu et al., 2025c). Transformer approaches learn timestep-dependent caching schedules or broadcast attention features at prescribed intervals (Ma et al., 2024a; Zhao et al., 2025). Online policies adapt refresh decisions to runtime observations of inexpensive input or shallow-feature changes (Liu et al., 2025a; Bu et al., 2026). At token granularity, ToCa and Jano allocate updates according to sensitivity or estimated convergence (Zou et al., 2025; Chen et al., 2026). Complementing these scheduling strategies, TaylorSeer extends reuse by forecasting future features through Taylor expansion (Liu et al., 2025b). HASTE connects these computation decisions to editable content through a hierarchy of output-layer scheduling and alpha-support sparsity. Skipped tokens advance using cached endpoints, while cached keys and values preserve cross-layer context during sparse execution.

6 Conclusion

We presented HASTE, a hierarchical adaptive reuse framework for accelerating text-to-layer generation. Our key observation is that redundancy in layered diffusion is structured at two levels: different layers converge at different rates, while computation within each layer is spatially concentrated around its alpha support. HASTE exploits this structure through adaptive layer selection, alpha-guided token selection, cached-endpoint extrapolation, and context-preserving sparse DiT execution. On Qwen-Image-Layered, HASTE achieves a $3.22\times$ sampling speedup while preserving decomposition quality, and it transfers to four additional T2L models with $1.52\times$–$4.26\times$ speedups.

AI USE STATEMENT

We used generative AI assistants for literature retrieval and summarization, polishing manuscript text, and preparing figures, tables, and LaTeX. AI assistants also supported experimental design, implementation and debugging of experiment code. The authors retain responsibility for the scientific decisions and the final content of this work, including its text, code, figures, and reported results.

ETHICS STATEMENT

This work aims to reduce the computational cost of decomposing images into editable RGBA layers. Our experiments use existing benchmark data and pretrained models. The method does not address biases or harmful content inherited from the underlying models. Applications should respect image ownership, privacy, and consent, and disclose manipulations when appropriate.

REPRODUCIBILITY STATEMENT

Section 3 describes the layer scheduling, alpha-support estimation, and sparse execution mechanisms. Section 4 and Appendix specifies the evaluated models, baselines, quality metrics, timing protocol, and implementation settings. For implementation and reproduction materials, we refer readers to our anonymous repository at `https://anonymous.4open.science/r/HASTE-0434`.

REFERENCES

Michael S. Albergo and Eric Vanden-Eijnden. Building Normalizing Flows with Stochastic Interpolants. In *International Conference on Learning Representations*, 2023.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Yibin Wang, Yuhang Zang, Dahua Lin, and Jiaqi Wang. DiCache: Let diffusion model determine its own cache. In *International Conference on Learning Representations*, 2026.

Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. PixArt-α: Fast Training of Diffusion Transformer for Photorealistic Text-to-Image Synthesis. In *International Conference on Learning Representations*, 2024a.

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*, 2024b.

Yuyang Chen, Linqian Zeng, Yijin Zhou, Hengjie Li, and Jidong Zhai. Jano: Adaptive diffusion generation with early-stage convergence awareness. *arXiv preprint arXiv:2603.00519*, 2026.

Tri Dao. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. In *International Conference on Learning Representations*, 2024.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations*, 2021.

Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling Rectified Flow Transformers for High-Resolution Image Synthesis. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235, pp. 12606–12633, 2024.

Junjia Huang, Pengxiang Yan, Jinhang Cai, Jiyang Liu, Zhao Wang, Yitong Wang, Xinglong Wu, and Guanbin Li. DreamLayer: Simultaneous Multi-Layer Generation via Diffusion Model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3357–3366, 2025.

Runhui Huang, Kaixin Cai, Jianhua Han, Xiaodan Liang, Renjing Pei, Guansong Lu, Songcen Xu, Wei Zhang, and Hang Xu. LayerDiff: Exploring Text-guided Multi-layered Composable Image Synthesis via Layer-Collaborative Diffusion Model. In *European Conference on Computer Vision*, 2024.

inclusionAI. Ming-Image-0.1-Design-Layer. Hugging Face model card, 2026. URL `https://huggingface.co/inclusionAI/Ming-Image-0.1-Design-Layer`.

Yueru Jia, Aosong Cheng, Yuhui Yuan, Chuke Wang, Ji Li, Huizhu Jia, and Shanghang Zhang. DesignEdit: Unify spatial-aware image editing via training-free inpainting with a multi-layered latent diffusion framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 3958–3966, 2025. doi: 10.1609/aaai.v39i4.32414.

Kumara Kahatapitiya, Haozhe Liu, Sen He, Ding Liu, Menglin Jia, Chenyang Zhang, Michael S. Ryoo, and Tian Xie. Adaptive Caching for Faster Video Generation with Diffusion Transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15240–15252, 2025.

Yoni Kasten, Dolev Ofri, Oliver Wang, and Tali Dekel. Layered Neural Atlases for Consistent Video Editing. *ACM Transactions on Graphics*, 40(6), 2021. doi: 10.1145/3478513.3480546.

Diederik P Kingma and Max Welling. Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*, 2014.

Anat Levin, Dani Lischinski, and Yair Weiss. A Closed-Form Solution to Natural Image Matting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(2):228–242, 2008. doi: 10.1109/TPAMI.2007.1177.

Senmao Li, Taihang Hu, Joost van de Weijer, Fahad Shahbaz Khan, Tao Liu, Linxuan Li, Shiqi Yang, Yaxing Wang, Ming-Ming Cheng, and Jian Yang. Faster diffusion: Rethinking the role of the encoder for diffusion model inference. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Zhimin Li, Jianwei Zhang, Qin Lin, Jiangfeng Xiong, Yanxin Long, Xinchi Deng, Yingfang Zhang, Xingchao Liu, Minbin Huang, Zedong Xiao, Dayou Chen, Jiajun He, Jiahao Li, Wenyue Li, Chen Zhang, Rongwei Quan, Jianxiang Lu, Jiabin Huang, Xiaoyan Yuan, Xiaoxiao Zheng, Yixuan Li, Jihong Zhang, Chao Zhang, Meng Chen, Jie Liu, Zheng Fang, Weiyan Wang, Jinbao Xue, Yangyu Tao, Jianchen Zhu, Kai Liu, Sihuan Lin, Yifu Sun, Yun Li, Dongdong Wang, Mingtao Chen, Zhichao Hu, Xiao Xiao, Yan Chen, Yuhong Liu, Wei Liu, Di Wang, Yong Yang, Jie Jiang, and Qinglin Lu. Hunyuan-DiT: A Powerful Multi-Resolution Diffusion Transformer with Fine-Grained Chinese Understanding. *arXiv preprint arXiv:2405.08748*, 2024b.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow Matching for Generative Modeling. In *International Conference on Learning Representations*, 2023.

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. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025a.

Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with TaylorSeers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 15853–15863, 2025b.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow Straight and Fast: Learning to Generate and Transfer Data with Rectified Flow. In *International Conference on Learning Representations*, 2023.

Xuewen Liu, Zhikai Li, and Qingyi Gu. CacheQuant: Comprehensively Accelerated Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 23269–23280, 2025c.

Zihao Liu, Zunnan Xu, Shi Shu, Jun Zhou, Ruicheng Zhang, Zhenchao Tang, and Xiu Li. Controllable layer decomposition for reversible multi-layer image generation. *arXiv preprint arXiv:2511.16249*, 2025d.

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.

Erika Lu, Forrester Cole, Tali Dekel, Andrew Zisserman, William T. Freeman, and Michael Rubinstein. Omnimatte: Associating Objects and Their Effects in Video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Zhengyao Lv, Chenyang Si, Junhao Song, Zhenyu Yang, Yu Qiao, Ziwei Liu, and Kwan-Yee K. Wong. FasterCache: Training-Free Video Diffusion Model Acceleration with High Quality. In *International Conference on Learning Representations*, 2025.

Xinyin Ma, Gongfan Fang, Michael Bi Mi, and Xinchao Wang. Learning-to-cache: Accelerating diffusion transformer via layer caching. In *Advances in Neural Information Processing Systems*, volume 37, 2024a.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15762–15772, 2024b.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

Thomas Porter and Tom Duff. Compositing Digital Images. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 253–259, 1984. doi: 10.1145/800031.808606.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-Resolution Image Synthesis with Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

Ciara Rowles, Reshinth Adithyan, Nikhil Pinnaparaju, Vikram Voleti, and Mark Boss. Stable-Layers: Fine-tuning image layer decomposition models with VLM-scored reinforcement learning. *arXiv preprint arXiv:2605.30257*, 2026.

Vishnu Sarukkai, Linden Li, Arden Ma, Christopher Ré, and Kayvon Fatahalian. Collage Diffusion. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 4208–4217, 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.

Xuan Shen, Zhao Song, Yufa Zhou, Bo Chen, Yanyu Li, Yifan Gong, Kai Zhang, Hao Tan, Jason Kuen, Henghui Ding, Zhihao Shu, Wei Niu, Pu Zhao, Yanzhi Wang, and Jiuxiang Gu. LazyDiT: Lazy Learning for the Acceleration of Diffusion Transformers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Byunggwan Son, Jeimin Jeon, Jeongwoo Choi, and Bumsub Ham. Relational feature caching for accelerating diffusion transformers. In *International Conference on Learning Representations*, 2026.

Tomoyuki Suzuki, Kang-Jun Liu, Naoto Inoue, and Kota Yamaguchi. LayerD: Decomposing raster graphic designs into layers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Petru-Daniel Tudosiu, Yongxin Yang, Shifeng Zhang, Fei Chen, Steven McDonagh, Gerasimos Lampouras, Ignacio Iacobacci, and Sarah Parisot. MULAN: A Multi Layer Annotated Dataset for Controllable Text-to-Image Generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Binhao Wang, Shihao Zhao, Bo Cheng, Qiuyu Ji, Yuhang Ma, Liebucha Wu, Shanyuan Liu, Dawei Leng, and Yuhui Yin. RevealLayer: Disentangling hidden and visible layers via occlusion-aware image decomposition. *arXiv preprint arXiv:2605.11818*, 2026.

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. doi: 10.1109/TIP.2003.819861.

Felix Wimbauer, Bichen Wu, Edgar Schoenfeld, Xiaoliang Dai, Ji Hou, Zijian He, Artsiom Sanakoyeu, Peizhao Zhang, Sam Tsai, Jonas Kohler, Christian Rupprecht, Daniel Cremers, Peter Vajda, and Jialiang Wang. Cache Me if You Can: Accelerating Diffusion Models through Block Caching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6211–6220, 2024.

Kota Yamaguchi. CanvasVAE: Learning to generate vector graphic documents. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021.

Jinrui Yang, Qing Liu, Yijun Li, Soo Ye Kim, Daniil Pakhomov, Mengwei Ren, Jianming Zhang, Zhe Lin, Cihang Xie, and Yuyin Zhou. Generative Image Layer Decomposition with Visual Effects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7643–7653, 2025.

Shengming Yin, Zekai Zhang, Zecheng Tang, Kaiyuan Gao, Xiao Xu, Kun Yan, Jiahao Li, Yilei Chen, Yuxiang Chen, Heung-Yeung Shum, Lionel M. Ni, Jingren Zhou, Junyang Lin, and Chenfei Wu. Qwen-Image-Layered: Towards inherent editability via layer decomposition. *arXiv preprint arXiv:2512.15603*, 2025.

Lvmin Zhang and Maneesh Agrawala. Transparent image layer diffusion using latent transparency. *ACM Transactions on Graphics*, 43(4), 2024. doi: 10.1145/3658150.

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.

Xinyang Zhang, Wentian Zhao, Xin Lu, and Jeff Chien. Text2Layer: Layered Image Generation using Latent Diffusion Model. *arXiv preprint arXiv:2307.09781*, 2023.

Xuanlei Zhao, Xiaolong Jin, Kai Wang, and Yang You. Real-time video generation with pyramid attention broadcast. In *International Conference on Learning Representations*, 2025.

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. In *International Conference on Learning Representations*, 2025.

Chang Zou, Shikang Zheng, Evelyn Zhang, Runlin Guo, Haohang Xu, Zhengyi Shi, Conghui He, Xuming Hu, and Linfeng Zhang. Rethinking Token-wise Feature Caching: Accelerating Diffusion Transformers with Dual Feature Caching. *IEEE Transactions on Image Processing*, 35:6211–6220, 2026.

APPENDIX CONTENTS

A Experimental Settings and Evaluation **15**
A.1 Crello data and layer decomposition 15
A.2 Quality metrics 15
A.3 Timing, memory, and computation 16
A.4 Baseline adaptations 17

B Implementation of Hierarchical Caching **17**
B.1 AlphaNet architecture and training 17
B.2 Support-domain scores and state updates 19
B.3 Endpoint reuse and sparse transformer execution 19
B.4 Computation complexity analysis 20

C Additional Quantitative Results **22**
C.1 Results across edit budgets 22
C.2 Layer count and per-layout variation 22
C.3 Admission-threshold operating points 23

D Analysis of Hierarchical Caching **25**
D.1 Layer-wise stabilization in a Full trajectory 25
D.2 Layer admission and actual token refresh 26
D.3 How priorities evolve between updates 27
D.4 Spatial redundancy within a layer 27
D.5 Temporal, spatial, and periodic-refresh components 31

E Generalization Across Models **33**
E.1 Model-specific execution and shared scheduling 33
E.2 Detailed transfer results 34

F Additional Qualitative Comparisons **34**
F.1 Selection and presentation 34
F.2 Comparison with caching baselines 34
F.3 Paired Full/**HASTE** decompositions 40
F.4 Cross-model paired decompositions 45

A Experimental Settings and Evaluation

A.1 Crello data and layer decomposition

We evaluate image decomposition using Crello 5.1.0 and the LayerD `CrelloV5RendererLayers` renderer. Text rendering is enabled, transparent elements are excluded, and inpainting is disabled. The requested output-layer count is obtained from the official occlusion-processing routine. Evaluation uses the rendered ground-truth layers and the official edit-aware matching procedure, following the protocol adopted by Qwen-Image-Layered.

The primary evaluation uses a fixed, GPU-memory-admitted subset of 1,000 layouts from the 1,971-layout test split. Inputs retain their original aspect ratios with a nominal 512-pixel short side. Integer truncation in the official preprocessing can produce a 511-pixel short side; this size is retained before inference padding. The same identities and input sizes are used for every compared method. We do not resize individual inputs to accommodate a particular caching method. Table 4 gives the layer-count composition of the evaluation cohorts and the subsets used for component ablation and operator profiling.

Table 4: Evaluation cohorts and their output-layer counts. The ablation and profiling subsets are drawn from the fixed 1,000-layout cohort. Stable-Layers uses a separate 300-layout transfer subset.

Cohort	Total	$N = 2$	$N = 3$	$N = 4$	$N = 5$	$N = 6$
Qwen primary evaluation	1000	290	340	200	150	20
CLD / RevealLayer / Ming	1000	290	340	200	150	20
Component ablations	500	150	170	100	70	10
HASTE threshold FLOPs	500	150	170	100	70	10
Baseline FLOPs	300	80	110	60	40	10
Stable-Layers transfer	300	70	70	70	70	20

All Qwen comparisons use the same prepared conditioning, caption, seed, output count, and initial noise. The public Qwen checkpoint is held fixed. Its identity has not been established to match the Crello-finetuned checkpoint associated with the published Qwen-Image-Layered benchmark numbers; our reported comparisons therefore use the paired Full runs produced in this evaluation.

A.2 Quality metrics

We distinguish task quality against ground-truth layers from fidelity to a model's own Full output. Table 5 summarizes the reference used by each metric. Both are relevant to caching: recomposition measures visible appearance, while layer-level evaluation also reflects the structure exposed to subsequent editing.

Table 5: Quality metrics and their comparison targets. Each layout receives equal weight in dataset aggregation.

Metric	Reference	Evaluation unit
LayerD αIoU / RGB L_1	Rendered ground truth	Edit-aware layer matching at budget k
Composite LPIPS / SSIM / PSNR	Same-model Full	Recomposition on a white background
Indexed RGBA RMSE	Same-model Full	Corresponding output slots, premultiplied RGBA
Indexed alpha IoU	Same-model Full	Corresponding alpha masks, threshold 0.5

Edit-aware decomposition quality. LayerD evaluates predicted layers against the rendered ground truth while allowing a limited number of editing operations. We report alpha overlap and RGB L_1 at an edit budget of five in the main tables. Appendix C.1 additionally reports the full budget range $k = 0, \ldots, 5$. If an evaluation path terminates earlier, its last value is retained at subsequent budgets. Output slots are not assumed to correspond directly to the original design elements.

Composite fidelity. We composite RGBA outputs in native back-to-front order on white, remove inference padding, and evaluate at the original evaluation resolution. LPIPS uses the official AlexNet weights, version 0.1, with RGB inputs scaled to $[-1, 1]$. SSIM uses an 11-pixel Gaussian window, standard deviation 1.5, population covariance, and a data range of one, averaged over RGB channels.

LPIPS and SSIM are averaged over layouts. For PSNR, we first compute an RGB mean-squared error for each layout and then convert the dataset mean:

$$\mathrm{PSNR} = -10 \log_{10} \left(\frac{1}{M} \sum_{j=1}^{M} \mathrm{MSE}_j \right). \tag{10}$$

This aggregation differs from averaging individual PSNR values. Full is its own reference, so its LPIPS, SSIM, and PSNR entries are shown as dashes.

Indexed-layer fidelity. For a layer with color C and opacity α, we evaluate the four-channel representation $[\alpha C, \alpha]$. Absolute and relative RMSE are computed for the corresponding output slot, averaged over layers within a layout, and then over layouts. Relative RMSE divides by the RMS of the Full layer. Hard alpha IoU thresholds both alpha maps at 0.5. We report either the mean over layers or the minimum within a layout, and specify the choice in the table caption. These indexed comparisons retain sensitivity to content moving between output slots, even when the recomposed image remains similar.

A.3 Timing, memory, and computation

The Qwen experiments use BF16 transformer computation, 50 denoising steps, normalized true CFG 4, and seed 777 on two 48-GB RTX A6000 GPUs. Each GPU holds a full transformer replica for one CFG branch. Branch communication uses the same host-memory relay for all methods. GPU-resident caching thus describes where inference caches are stored; it does not imply the absence of all host communication.

Sampling latency includes online observation, support prediction, scheduling, cache access, transformer execution, and any in-sampling checks. It excludes model loading, input preparation, and final RGBA decoding. Dataset speedup is the ratio of paired sampling-time totals,

$$\mathrm{Speedup} = \frac{\sum_j T_j^{\mathrm{Full}}}{\sum_j T_j^{\mathrm{HASTE}}}. \tag{11}$$

Per-layout speedups are used only in distribution plots and visual examples. They are not averaged to obtain the headline speedup. Each method runs in an independent process using the shared prepared input.

The stored runs expose both accumulated step timers and a complete denoising-loop wall timer. Table 6 makes their relationship explicit. For PAB and ToCa, the main table retains the original accumulated step timings, whereas the loop timer additionally includes work between those intervals. The layer-count breakdown in this appendix uses loop wall timers for every method. This gives PAB and ToCa mean loop latencies of 81.10 and 50.80 seconds, respectively; the generated outputs and quality measurements are identical under either timing aggregation.

Table 6: Recorded timing scopes on the fixed 1,000 layouts. The main table retains accumulated step timers for PAB and ToCa; the complete denoising-loop timer also covers time between those intervals. Grouped appendix results consistently use the complete loop timer.

Method	Main-table time (s)	Loop time (s)	Difference (s)	Loop speedup↑
Full	99.03	99.03	0.00	1.00
PAB	80.85	81.10	0.24	1.22
ToCa	49.62	50.80	1.17	1.95
Jano	52.55	52.55	0.00	1.88
HASTE	30.72	30.72	0.00	3.22

ToCa uses host-backed caches for 170 of the 1,000 layouts after GPU cache allocation fails. Actual cache-transfer time remains in its sampling latency; the inputs and inference configuration are preserved. All other methods in the main comparison retain GPU-resident caches. The accompanying execution records retain failed attempts separately from successful-run timing. A distinct post-loading latency measure includes setup, latent serialization, and final decoding; it is not substituted for sampling latency in the comparison tables.

Executed FLOPs count matrix multiplication, attention QK/AV products, and convolutions during sampling, including both CFG branches and applicable support/scoring operations. A fused multiply–add counts as two operations. Final decoding, normalization, reductions, nonlinearities, and complex rotary operations are excluded. Baseline profiling uses 300 layouts; the three **HASTE** thresholds use the same 500-layout subset (Table 4). These subset means describe their respective profiled cohorts. Profiling is separate from normal timing. All 1,500 **HASTE** profiling trajectories reproduce the corresponding normal outputs and query schedules. Appendix B.4 derives the corresponding leading-order costs of full execution, hierarchical selection, and cache reuse.

A.4 Baseline adaptations

The baselines retain their respective reuse mechanisms while operating within the Qwen layered transformer. Table 7 summarizes their settings. Scoring, cache management, and transfers are included in the sampling measurements.

Table 7: Baseline mechanisms and settings used in the reported Qwen comparison.

Method	Reused computation	Scheduling configuration
PAB	Joint-attention and supported MLP outputs	Open-Sora broadcast schedule; MLP timestep keys mapped to the nearest sampled timestep
ToCa	Token-level module features	Refresh interval 3; base fresh-token ratio 0.30; PixArt scoring adapted to 60 blocks and per-frame token grids
Jano	Globally selected image tokens and cached context	Six warmup predictions; score thresholds 0.2 / 0.6; refresh periods 6 / 3 / 1; four final full steps

PAB. Qwen uses joint attention rather than separate temporal and cross-attention modules. We broadcast the joint-attention output and supported MLP outputs; the absent attention types are not counted as separate broadcast operations.

ToCa. Token scoring follows the adapted PixArt configuration. Its attention-score pass executes within the measured sampling loop. The per-frame spatial grid is retained when forming token scores for the layered sequence.

Jano. Per-token distance-change scores are computed from six initial guided predictions over the complete image-token sequence: Aux, every editable output layer, and Ref. Scores are min–max normalized globally and assigned directly to static, moderate, and active levels. Tokens are not first partitioned into an arbitrary number of equal-sized layer groups. The three levels refresh at periods of six, three, and one steps, respectively. Analysis occurs once after warmup; four final steps use full computation. Ref follows the same token-level query/cache schedule, while its conditioning latent remains fixed and is never advanced by the sampler. **HASTE** and Jano store inactive K/V in FP8 and perform backbone computation in BF16. All Jano tables in this appendix use this all-frame global-token configuration.

B Implementation of Hierarchical Caching

B.1 AlphaNet architecture and training

AlphaNet predicts opacity on the packed token grid from cached clean endpoints. For the Qwen latent representation, each packed token has 64 channels. The predictor reshapes the sequence into its two-dimensional grid, applies a small convolutional network, and returns one scalar in $[0, 1]$ per token. Table 8 details the network architecture, and Table 9 lists the training settings. The transformer and RGBA VAE remain frozen.

Training examples cover intermediate endpoints at all 50 denoising steps and multiple requested layer counts. For each endpoint, the frozen VAE supplies the alpha target. The four alpha subpositions associated with a packed token are averaged before supervision. This target estimates the token's own opacity, including regions occluded by higher layers; it is not the token's final visible contribution to the recomposed image.

We assess prediction using four leave-one-content-out splits. All layer counts and timesteps of the held-out content remain outside training. The predictor used in the caching experiments is

Table 8: **AlphaNet network architecture.** $H_t \times W_t$ is the packed token grid, with $S = H_t W_t$; batch dimensions are omitted. All convolutions use stride 1, with padding 1 for 3×3 kernels and 0 for 1×1 kernels. GN(8) denotes GroupNorm with eight groups. Spatial resolution is preserved.

Stage	Operation	Output shape	Parameters
Input	Reshape packed endpoints	$64 \times H_t \times W_t$	0
Stem	3×3 Conv, $64 \to 48$	$48 \times H_t \times W_t$	27,696
Hidden block $\times 3$	GN(8) $\to$ SiLU $\to$ 3×3 Conv, $48 \to 48$	$48 \times H_t \times W_t$	$3 \times 20{,}880$
Output head	GN(8) $\to$ SiLU $\to$ 1×1 Conv, $48 \to 1$	$1 \times H_t \times W_t$	145
Output	Sigmoid; flatten spatial grid	$S \times 1$	0
Total trainable parameters			**90,481**

Table 9: AlphaNet training configuration.

Item	Configuration
Supervision	Exact decoded alpha, averaged over four latent subpositions per token
Training examples	5,100 layer–step examples from 18 trajectories and four development contents
Optimizer	AdamW; learning rate 2×10^{-3}; weight decay 10^{-4}
Training schedule	60 epochs; cosine learning-rate decay; batch size 64; seed 0
Loss	Mean absolute error plus $0.1\times$ binary cross-entropy with soft alpha targets

subsequently trained on all four development contents. Table 10 reports held-out opacity error and support classification, and Fig. 9 separates error by noise level. The training and evaluation manifests preserve content identities, trajectory hashes, and supervision settings.

Table 10: Content-held-out AlphaNet validation. Each split excludes all trajectories of one content from training. False-freeze rate is the fraction of predicted-frozen tokens whose target alpha is at least 0.05; recall is measured over target-support tokens.

Held-out content	Layer–step samples	$L_1 \downarrow$	AUROC↑	Recall↑	False freeze↓
Halloween	1650	0.0113	0.9980	0.9733	0.0128
Halloween city	900	0.0103	0.9987	0.9703	0.0105
Temple	1650	0.0242	0.9789	0.8974	0.0753
Flat illustration	900	0.0282	0.9811	0.9086	0.0705

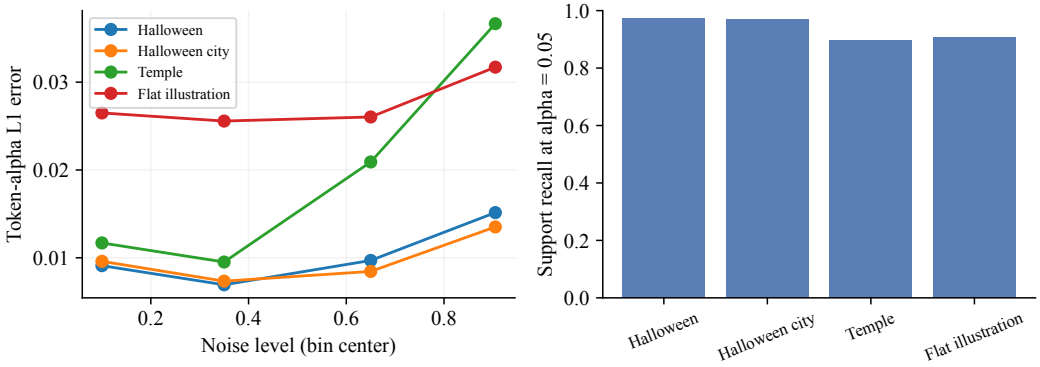

Figure 9: **Content-held-out alpha prediction.** Left: opacity prediction error in noise-level bins for each held-out content. Right: recall of target support at alpha threshold 0.05. These are predictor-validation measurements, separate from the Crello caching benchmark.

The held-out results span alpha L_1 errors of 0.0103–0.0282 and support AUROC of 0.9789–0.9987. Support recall ranges from 0.8974 to 0.9733. Errors at early, high-noise endpoints motivate delaying spatial restriction and adding a one-token boundary margin. The predictor supplies an inexpensive

online mask; periodic refresh gives positions outside that mask another opportunity to receive fresh computation.

B.2 SUPPORT-DOMAIN SCORES AND STATE UPDATES

Table 11 collects the settings shared across Qwen output-layer counts. The scheduler computes support before layer admission and uses that same support for score reduction, selected-token execution, and endpoint calibration. This alignment connects a layer's priority to the positions that its admission can actually update.

Table 11: **HASTE** configuration for the 50-step Qwen experiments. These settings are shared by every evaluated output-layer count.

Component	Setting
Layer priority	$P_{t,i} = \kappa_{t,i} B_{t,i}$
Admission threshold	$\tau = 0.16$; additional operating points 0.08 and 0.20
EMA new-feedback weight	0.2
Sensitivity update	First valid feedback initializes; target clipped to $[0.01, 100]$
Endpoint normalization floor	0.2
Minimum calibration signal	$B_{t,i} \geq 10^{-5}$; ratio stabilizer 10^{-8}
Alpha support	Threshold 0.05; 3×3 dilation
Spatial-filter start	Step 6, zero-based
Warmup / refinement	Six initial full steps / one final full step
Periodic full refresh	Every ten steps in the caching schedule
Ref / Aux timer setting	Age threshold 9; maximum timer-only gap 10 steps
Alpha monitoring	Check after five non-full steps; confirm at the next check
Reuse / cache precision	Clean endpoints / FP8 inactive K/V; BF16 backbone

Accumulation while a layer is skipped. The online observation is obtained before the expensive transformer stack. Token-level feature changes are accumulated across steps, with the current noise level weighting each increment. Consequently, a layer can acquire a larger score while its cached endpoint remains unchanged. No fixed number of layers is required to refresh: the admitted set can contain all, some, or none of the editable layers.

Calibration on matching positions. Once an admitted layer is evaluated, the endpoint displacement is measured over the same scoring support. The feedback target divides this normalized displacement by the accumulated observation on that support. Calibration is performed only when every scoring-support position was freshly evaluated and the accumulated signal is at least 10^{-5}. The first valid observation initializes the sensitivity; subsequent observations use EMA with new-value weight 0.2. The ratio target is clipped to $[0.01, 100]$ before the EMA update.

Resetting only executed tokens. After an update, the accumulator is reset only at positions that received fresh computation. Skipped positions keep their state. Positions outside the current support are excluded from the current layer score, so their unresolved variation cannot repeatedly trigger admission while the spatial selector continues to exclude them. If a later support mask includes those positions, their accumulated state becomes eligible for scoring. The reported main configuration uses periodic full refresh and the existing monitor, without an additional independent outside-support probing mechanism.

B.3 ENDPOINT REUSE AND SPARSE TRANSFORMER EXECUTION

The endpoint cache and K/V cache play different roles. Endpoint reuse advances skipped generated tokens through the diffusion trajectory. K/V reuse supplies attention context for the tokens that are freshly evaluated. Neither operation removes the skipped layer from the model's cross-layer context.

For a skipped generated token, Eq. 7 can be written as

$$z_{t+1} = \overline{x}_t + \frac{\sigma_{t+1}}{\sigma_t}(z_t - \overline{x}_t), \qquad \sigma_t > 0. \tag{12}$$

Thus the latent continues toward its cached clean prediction even when its transformer evaluation is skipped. If the same endpoint is reused over several steps, the noise-level ratios telescope. This

distinguishes endpoint reuse from holding the noisy latent constant. It also differs from residual reuse, which adds a cached network residual to a current network input; **HASTE**'s reported runs use the endpoint rule above.

Fresh image tokens and text tokens issue queries during sparse execution. Keys and values for skipped image tokens are read from persistent per-block caches. Fresh positions overwrite their corresponding cache entries; skipped positions retain theirs. The attention key sequence therefore still spans the full layered image context, while query projections, attention queries, and tokenwise transformer computation are restricted to the executed positions.

The auxiliary generated frame and static reference frame have separate roles. Aux follows generated-latent evolution and can use endpoint reuse. Ref remains fixed conditioning and is never advanced by the ODE. Its refresh decision determines whether its transformer representations and cache entries are recomputed. Their timer setting of nine is an age threshold: without an earlier full update, it corresponds to a ten-step refresh gap. Full steps refresh both frames together with all editable tokens.

Algorithm 1 summarizes the execution order. All monitoring and scheduling work is included in measured sampling time.

Algorithm 1 **HASTE** sampling with support-aligned state updates

Require: Initial latents, noise schedule, conditioning, threshold τ
Initialize endpoint caches, K/V caches, sensitivities, and token accumulators
for each denoising step t **do**
  Observe inexpensive token features and accumulate their changes
  Estimate alpha support from cached endpoints; use full support before initialization
  if warmup, periodic refresh, refinement, or a triggered full-update guard **then**
    Select all image tokens for fresh computation
  else
    Compute $P_{t,i} = \kappa_{t,i}\,\mathrm{RMS}_{\mathcal{S}_{t,i}}(G_{t,i})$
    Select layers satisfying $P_{t,i} > \tau$ and their support tokens
    Add Ref and Aux tokens when their refresh timers request an update
  end if
  if the fresh image-token set is nonempty **then**
    Evaluate fresh queries against fresh and cached full-sequence K/V
    Update fresh endpoints and K/V; calibrate on fully covered scoring supports
    Reset accumulators only at the executed output-token positions
  end if
  Advance generated latents using fresh velocities or cached endpoints
  Update refresh timers and monitoring state
end for
Decode the final editable RGBA layers

B.4 Computation complexity analysis

Following the operation-wise analysis of Zou et al. (2025), we separate the dominant transformer computation from the cost of deciding what to refresh. We consider one sample and one guidance branch of Qwen-Image-Layered. Let N be the number of editable layers, S the tokens per frame, and $M = (N+2)S$ the image-token count, including Aux and Ref. The joint sequence has length $L = M + n_{\mathrm{txt}}$. We use $D = d_h$ for the total hidden width across all attention heads, $D_{\mathrm{ff}} = rD$ for the FFN intermediate width, and B for the number of transformer blocks. A multiply–add counts as two FLOPs. The expressions below count the leading matrix products, consistently with the profiling convention in Appendix A.3.

B.4.1 Main computations

Complexity of joint attention. Full execution projects every image and text token into queries, keys, and values, computes attention over the complete sequence, and applies an output projection. Its per-block cost is

$$\begin{aligned} C_{\mathrm{attn}}^{\mathrm{Full}} &\simeq 2LD(3D) + 2L^2D + 2L^2D + 2LD^2 \\ &= 8LD^2 + 4L^2D = \mathcal{O}(LD^2 + L^2D). \end{aligned} \tag{13}$$

The two quadratic terms correspond to $QK^\top$ and multiplication by V. Image and text projections have separate weights but contribute the same per-token cost at width D. Qwen uses joint attention, so text conditioning is already included in this expression.

At step t, **HASTE** freshly evaluates $m_t = |\mathcal{F}_t|$ image tokens. Whenever $m_t > 0$, text tokens are also evaluated, giving $q_t = m_t + n_{\text{txt}}$ query rows. Fresh image and text rows receive new Q/K/V projections, while cached K/V supply the remaining image context. The key/value length stays at L, and the cost becomes

$$C_{\text{attn}}^{\text{HASTE}}(t) \simeq 8q_tD^2 + 4q_tLD = \mathcal{O}(q_tD^2 + q_tLD). \tag{14}$$

Thus attention becomes rectangular: its query dimension shrinks with the fresh set while every query retains access to all layers. The attention products scale by q_t/L, rather than $(q_t/L)^2$, because their key/value dimension remains unchanged. Softmax contributes an additional $\mathcal{O}(Hq_tL)$ scalar operations for H heads, with the same query dependence; it is excluded from the matrix-product FLOPs above.

Complexity of the feed-forward network. The two FFN linear transformations map D channels to rD channels and back. They therefore require

$$C_{\text{FFN}}^{\text{Full}} \simeq 4rLD^2, \qquad C_{\text{FFN}}^{\text{HASTE}}(t) \simeq 4rq_tD^2. \tag{15}$$

For an expansion ratio of four, these become $16LD^2$ and $16q_tD^2$. Both fresh image rows and text rows pass through their respective FFNs. Table 12 summarizes the resulting computation.

Table 12: **Leading per-block computation.** The **HASTE** column describes an executed step with $m_t > 0$. All entries count matrix-product FLOPs, with two operations per multiply–add. Cached image K/V retain the full context length L while only q_t rows receive fresh transformer computation.

Operation	Full	**HASTE**	Retained fraction
Q/K/V and output projections	$8LD^2$	$8q_tD^2$	q_t/L
Attention products	$4L^2D$	$4q_tLD$	q_t/L
Feed-forward linear layers	$4rLD^2$	$4rq_tD^2$	q_t/L

From hierarchical selection to executed tokens. Let $a_{t,i} \in \{0,1\}$ indicate admission of editable layer i and $u_{t,i} = |\mathcal{S}_{t,i}|/S$ its support fraction. On a selective step,

$$m_t = S\sum_{i=1}^{N} a_{t,i}u_{t,i} + Sa_t^{\text{Aux}} + Sa_t^{\text{Ref}}, \tag{16}$$

where a_t^{Aux} and a_t^{Ref} indicate whole-frame refreshes. Temporal selection removes the contribution of an unselected layer, and spatial selection reduces the contribution of an admitted layer to its support. Ref and Aux are included in the executed-token budget. If $m_t = 0$, the implementation bypasses the entire transformer stack, including text computation. We define $q_t = 0$ for this case. Full refreshes set $m_t = M$ and $q_t = L$.

B.4.2 Computation costs from scheduling and cache reuse

Online observation and layer scores. The scheduler observes the time-modulated input of the first block before attention or FFN execution. The image input projection costs $2Md_zD$ FLOPs, where $d_z = 64$ is the packed latent width. Input normalization, modulation, and channel-wise feature differences take $\mathcal{O}(MD)$ work, while the shared timestep embedding and modulation projections contribute $\mathcal{O}(D^2)$. Hence the observation costs $\mathcal{O}(Md_zD + MD + D^2)$ per branch and step, without evaluating a transformer block. Both guidance branches are observed, and their token changes are combined by a pointwise maximum. Accumulating $G_{t,i,p}$ and reducing it over all layer supports costs $\mathcal{O}(NS)$, followed by $\mathcal{O}(N)$ priority comparisons. Independent thresholding requires no global token sort.

Alpha-support prediction and sensitivity calibration. AlphaNet operates on the token grid with a fixed convolutional width $C = 48$. Its four 3×3 convolutions and final 1×1 projection require

$$C_\alpha \simeq 2NS\big(9d_zC + 27C^2 + C\big) = 179{,}808\,NS \tag{17}$$

FLOPs per evaluation over all editable layers. This cost is linear in the number of image tokens and is incurred once per support prediction, outside the B transformer blocks. Thresholding and fixed 3×3 dilation add $\mathcal{O}(NS)$ work. After a refresh, endpoint differences over the executed support cost $\mathcal{O}(m_t d_z)$, and the layer-wise sensitivity EMA costs $\mathcal{O}(N)$. These updates reuse the freshly computed endpoint and require no extra transformer evaluation.

Endpoint and K/V maintenance. Advancing skipped generated tokens from cached endpoints takes $\mathcal{O}((N+1)Sd_z)$ work per step. During an executed transformer step, reading, converting, and assembling image K/V require $\mathcal{O}(BMD)$ work, and updating fresh cache entries requires $\mathcal{O}(Bm_tD)$. Persistent K/V storage is $\Theta(BMD)$ per branch. FP8 storage reduces its byte footprint while preserving all image positions. Endpoint storage is $\mathcal{O}((N+1)Sd_z)$, the previous observed features occupy $\mathcal{O}(NSD)$ per branch, and token accumulators and support masks occupy $\mathcal{O}(NS)$. K/V access is bypassed on zero-block steps.

Total computation across denoising. Let T_{full} count steps with $m_t = M$, including warmup, periodic refreshes, triggered full updates, and refinement. Let $\mathcal{T}_{\text{sel}}$ contain the remaining steps with $0 < m_t < M$. For fixed sequence lengths and model widths, Eqs. 13–15 sum to $B[(8+4r)q_tD^2 + 4q_tLD]$ across the stack at step t. The retained fraction of dominant transformer FLOPs over the trajectory is therefore

$$\frac{C^{\text{stack}}_{\text{HASTE}}}{C^{\text{stack}}_{\text{Full}}} \simeq \frac{1}{T}\sum_{t=1}^{T}\frac{q_t}{L} = \frac{T_{\text{full}}}{T} + \frac{1}{T}\sum_{t\in\mathcal{T}_{\text{sel}}}\frac{m_t + n_{\text{txt}}}{M + n_{\text{txt}}}. \tag{18}$$

The formula applies to each guidance branch; total FLOPs sum the two branches. For image-dominated sequences, a selective step refreshing 10% of image tokens uses approximately 10% of the leading stack computation. Zero-block steps remove that stack cost entirely, while full refreshes contribute their full share. This connects the two scheduling levels in Eq. 16 directly to computation savings.

Actual sampling also includes input/output projections, online scheduling, cache maintenance, sampler updates, and any alpha-monitoring decodes. These costs, together with memory traffic and execution overhead, determine wall-clock acceleration. In the Qwen Crello-100 runs, the recorded observation, support-prediction, and mask-assembly intervals total 0.764 seconds per image, or 2.49% of **HASTE**'s 30.72-second sampling time. Other operations remain in their enclosing execution intervals. Figure 8 reports the measured time distribution alongside this operation-level analysis.

C Additional Quantitative Results

C.1 Results across edit budgets

Figure 10 and Tables 13 and 14 extend the main comparison across every edit budget. All methods are evaluated against the same rendered ground truth. Increasing the budget lets the evaluator account for differences between generated layer groupings and the original design structure. The curves make this dependence explicit rather than interpreting the $k = 5$ result as a direct slotwise match.

Table 13: Ground-truth αIoU↑ across LayerD edit budgets on the same 1000 layouts. An exhausted edit path retains its last value. Jano uses global token scheduling over all image frames.

Method	$k=0$	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$
Full	0.7168	0.7777	0.8632	0.8718	0.8862	0.8874
PAB	0.7176	0.7792	0.8627	0.8711	0.8855	0.8867
ToCa	0.7099	0.7704	0.8538	0.8631	0.8796	0.8809
Jano	0.7180	0.7804	0.8638	0.8715	0.8844	0.8864
HASTE	0.7146	0.7762	0.8628	0.8706	0.8845	0.8871

C.2 Layer count and per-layout variation

Table 15 groups the primary comparison by output-layer count, using the complete-loop timing scope in Table 6. **HASTE** uses the same threshold and refresh settings throughout, rather than a

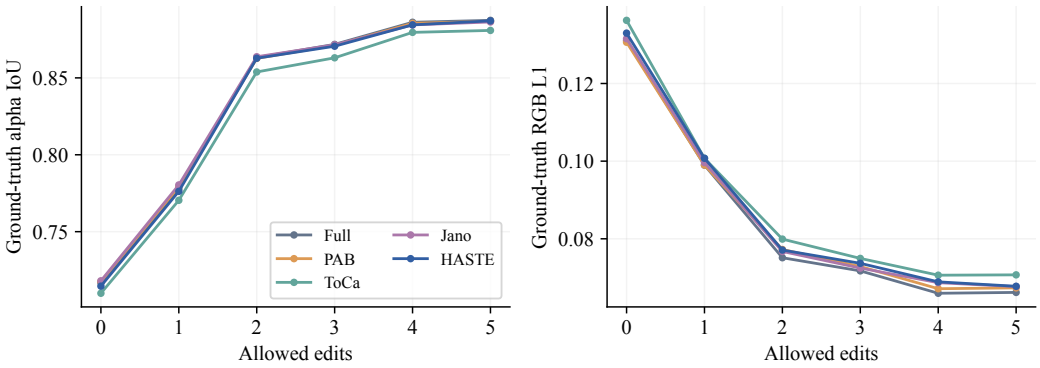

Figure 10: **Decomposition quality across edit budgets.** Each point averages the same 1000 Crello layouts. **HASTE** uses $\tau = 0.16$. The horizontal axis is the allowed evaluation-edit budget, not a diffusion timestep.

Table 14: Ground-truth RGB $L_1 \downarrow$ across LayerD edit budgets on the same 1000 layouts. An exhausted edit path retains its last value. Jano uses global token scheduling over all image frames.

Method	$k = 0$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
Full	0.1311	0.0990	0.0751	0.0717	0.0660	0.0662
PAB	0.1306	0.0990	0.0772	0.0729	0.0671	0.0674
ToCa	0.1363	0.1007	0.0799	0.0749	0.0706	0.0707
Jano	0.1315	0.0996	0.0767	0.0725	0.0687	0.0677
HASTE	0.1330	0.1007	0.0771	0.0737	0.0689	0.0678

separate configuration for each N. This grouping exposes the joint effect of sequence size and content composition. The $N = 6$ group contains 20 layouts.

Figure 11 chooses 100 layouts for observations. Composite similarity and indexed-layer error describe complementary outcomes: an image can retain its visible layout while some content changes output slot or the opacity of an individual layer changes. Table 16 therefore reports both forms of fidelity alongside the task-level results. The scatter plots display their variation without filtering by a quality or speed threshold.

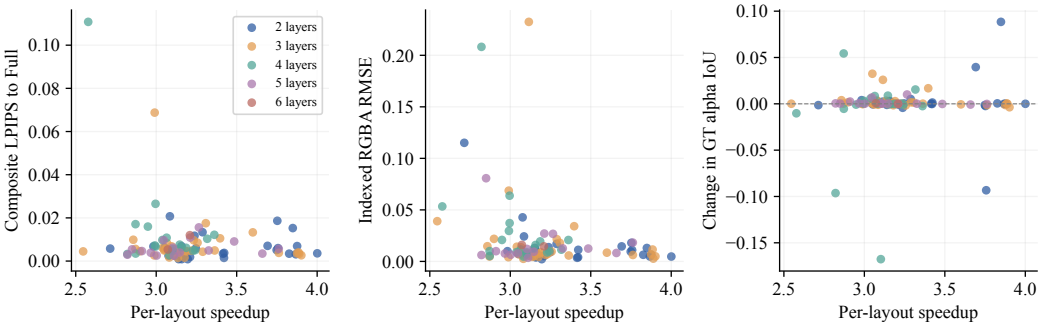

Figure 11: **Per-layout efficiency and quality of HASTE.** All chosen 100 layouts are shown at $\tau = 0.16$, colored by output-layer count. The right panel reports the paired change in ground-truth alpha IoU; zero indicates agreement in the task metric, rather than pixel-identical layers.

C.3 Admission-threshold operating points

The admission threshold controls how much accumulated change is tolerated before another layer update. Table 17 and Fig. 12 compare thresholds 0.08, 0.16, and 0.20 on the same 100 layouts. Warmup, refinement, support filtering, calibration, and periodic refresh remain unchanged. The corresponding overall speedups are $2.75\times$, $3.22\times$, and $3.48\times$.

Table 15: Results grouped by output-layer count. Timing uses complete denoising-loop wall timers for every method; speedup is the ratio of paired time totals. The six-layer group contains 20 layouts.

N	Method	n	Time (s)	Speedup↑	αIoU↑	RGB L_1 ↓
2	Full	290	71.08	1.00	0.8867	0.0700
2	PAB	290	59.01	1.20	0.8877	0.0699
2	ToCa	290	37.06	1.92	0.8776	0.0706
2	Jano	290	39.01	1.82	0.8895	0.0700
2	**HASTE**	290	20.70	3.43	0.8883	0.0715
3	Full	340	100.23	1.00	0.9252	0.0388
3	PAB	340	82.04	1.22	0.9270	0.0388
3	ToCa	340	51.37	1.95	0.9184	0.0460
3	Jano	340	52.85	1.90	0.9268	0.0386
3	**HASTE**	340	30.93	3.24	0.9277	0.0388
4	Full	200	106.62	1.00	0.8554	0.0962
4	PAB	200	87.18	1.22	0.8467	0.1026
4	ToCa	200	54.67	1.95	0.8516	0.0984
4	Jano	200	56.98	1.87	0.8413	0.1029
4	**HASTE**	200	34.71	3.07	0.8461	0.0983
5	Full	150	130.20	1.00	0.8688	0.0745
5	PAB	150	105.67	1.23	0.8703	0.0740
5	ToCa	150	65.53	1.99	0.8644	0.0826
5	Jano	150	67.47	1.93	0.8717	0.0760
5	**HASTE**	150	40.98	3.18	0.8702	0.0793
6	Full	20	174.43	1.00	0.7154	0.1140
6	PAB	20	140.26	1.24	0.7129	0.1141
6	ToCa	20	90.99	1.92	0.7056	0.1262
6	Jano	20	87.92	1.98	0.7141	0.1138
6	**HASTE**	20	55.72	3.13	0.7157	0.1140

Table 16: Indexed-layer fidelity and composite fidelity to the paired Full output on 1,000 layouts. RGBA errors use premultiplied RGB and alpha. Minimum alpha IoU is computed within each layout, then averaged.

Method	RGBA abs.↓	RGBA rel.↓	Min. αIoU↑	LPIPS↓	SSIM↑	PSNR↑
Full	0.0000	0.0000	1.0000	–	–	–
PAB	0.0075	0.0506	0.9606	0.0038	0.9911	39.36
ToCa	0.0255	0.1251	0.8746	0.0451	0.9516	28.77
Jano	0.0140	0.0685	0.9299	0.0055	0.9905	37.60
HASTE	0.0189	0.0915	0.9121	0.0081	0.9872	35.54

The quality curves need not be monotonic in every layer-count group: changing the admission decision changes subsequent endpoints and attention context. We consequently report each measured operating point, together with both task quality and fidelity, rather than treating the threshold as an exact quality-error bound. The matched operator profile decreases from 4.871 to 4.299 to 4.041 PFLOPs on its 50-layout cohort as the threshold increases.

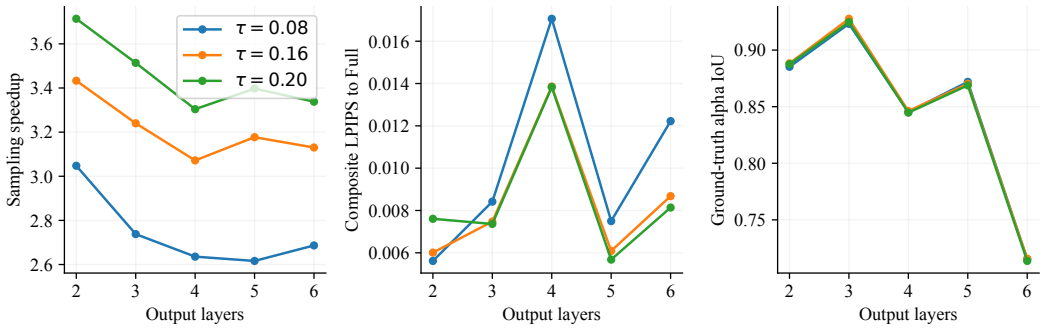

Figure 12: **Admission-threshold trade-offs across layer counts.** The same 100-layout cohort is used at every threshold. The small six-layer group is included without extrapolating its trend to other contents.

Table 17: Admission-threshold comparison on the same 100 layouts. Every other scheduling setting is fixed. Quality is evaluated at the native Crello resolution.

N	τ	Speedup↑	Time (s)	GT αIoU↑	LPIPS↓	SSIM↑
2	0.08	3.05	23.32	0.8852	0.0056	0.9900
2	0.16	3.43	20.70	0.8883	0.0060	0.9898
2	0.20	3.71	19.14	0.8876	0.0076	0.9892
3	0.08	2.74	36.61	0.9232	0.0084	0.9869
3	0.16	3.24	30.93	0.9277	0.0075	0.9884
3	0.20	3.51	28.52	0.9248	0.0074	0.9887
4	0.08	2.64	40.45	0.8454	0.0171	0.9786
4	0.16	3.07	34.71	0.8461	0.0139	0.9831
4	0.20	3.30	32.27	0.8447	0.0138	0.9832
5	0.08	2.62	49.76	0.8719	0.0075	0.9788
5	0.16	3.18	40.98	0.8702	0.0061	0.9858
5	0.20	3.40	38.32	0.8688	0.0057	0.9860
6	0.08	2.69	64.93	0.7153	0.0122	0.9778
6	0.16	3.13	55.72	0.7157	0.0087	0.9824
6	0.20	3.34	52.26	0.7137	0.0081	0.9828

D Analysis of Hierarchical Caching

D.1 Layer-wise stabilization in a Full trajectory

Figure 13 expands the temporal observation behind the hierarchical schedule. For each layer, we compare its intermediate endpoint against the final latent, using one RMS normalization shared across the trajectory. Stabilization is the earliest step after which this relative error remains below a chosen tolerance. It is measured retrospectively on a Full trajectory and is distinct from the online admission threshold.

At an 8% tolerance, the six layers in the displayed order stabilize at steps 3, 27, 21, 17, 24, and 31, giving a 28-step spread within the same sampling trajectory. This example was selected from archived trajectories to illustrate a pronounced temporal difference. Table 18 shows its measured stabilization times across tolerances, retaining the same token domain and normalization. Whole-layer RMS

stabilization and the median token stabilization time need not coincide because the former aggregates error before thresholding and the latter aggregates threshold-crossing times.

Table 18: Stabilization tolerance on the same illustrative six-layer Full trajectory. Entries are zero-based steps at which whole-layer relative RMS error first remains below the tolerance. The tolerance does not change any sampling or caching configuration.

Tolerance	L0	L1	L2	L3	L4	L5	Step spread
5%	36	42	40	38	37	44	8
6%	24	36	34	32	31	41	17
7%	6	28	22	24	25	38	32
8%	3	27	21	17	24	31	28
9%	3	14	21	17	24	29	26
10%	2	14	20	16	23	29	27

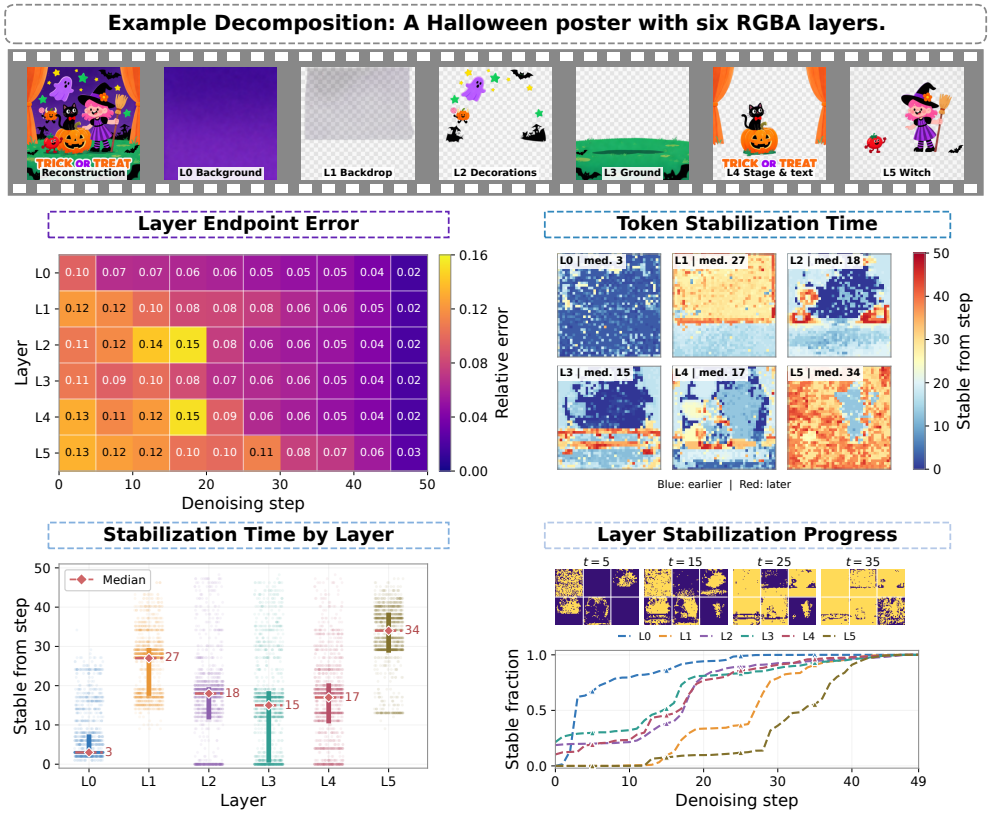

Figure 13: **Different stabilization times within one decomposition.** The archived 50-step, six-layer trajectory uses seed 1234. All editable tokens are included; the endpoint-error normalization is shared across layers and uses the final generated latent, including Aux. Token-level stabilization maps and whole-layer progress use the same 8% tolerance.

D.2 Layer admission and actual token refresh

Table 19 and Fig. 14 separate four execution states: dense full updates, sparse updates that select all output layers, updates selecting a proper subset of output layers, and steps that select no output layer. This distinction separates the temporal decision from the spatial amount of executed work.

Across random 100 trajectories, **HASTE** produces 603 partial-layer steps, and 85 layouts contain at least one such step. It also skips the transformer stack on 3,205 of the 5,000 sampling steps. These observations show that the measured execution combines selective layer updates with whole-step reuse. The number of partial-layer steps varies with the layout and output-layer count.

Table 19: Mean number of steps by execution state over the 50-step trajectory, grouped by output-layer count. All-layer sparse steps select every output layer but retain token filtering. Zero-output steps can still refresh Ref or Aux; zero-block steps skip the transformer stack.

N	n	Dense full	All-layer sparse	Partial layer	Zero output	Zero block
2	29	11.03	1.28	2.59	35.10	35.10
3	34	11.35	0.56	5.97	32.12	32.12
4	20	11.60	0.20	8.25	29.95	29.95
5	15	11.40	0.00	9.13	29.47	29.47
6	2	11.50	0.00	11.50	27.00	27.00

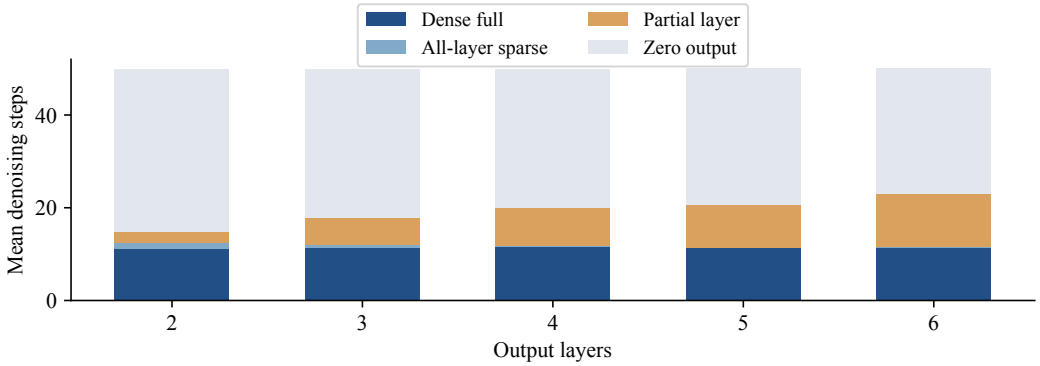

Figure 14: **Execution-state distribution.** Bars sum to 50 steps and average layouts within each layer-count group. Sparse all-layer admission can still save computation through token filtering.

Figures 15–17 show the actual token fractions for the 18 Qwen examples used in the qualitative gallery. A white cell means that the layer is reused, a dark-blue cell means that all its tokens are refreshed, and intermediate shades indicate partial spatial execution. Ref and Aux are displayed separately. The broad initial bands and final bands correspond to full evaluation; intermediate updates reveal content-dependent differences among layers.

D.3 How priorities evolve between updates

Figure 18 plots the pre-update priorities of four recorded examples. Within each of the two-, three-, four-, and five-layer groups in the gallery, the example with the most partial-layer steps is shown to make the admission mechanism visible. Markers indicate non-full steps on which the corresponding layer is selected. The common threshold does not impose a common refresh time: accumulated observations, support masks, and calibrated sensitivities differ across layers. Periodic or guard-triggered full steps can also update a layer independently of a threshold crossing.

D.4 Spatial redundancy within a layer

Spatial selection uses a layer's own alpha support, including opaque content that may be hidden by a higher layer in the final composite. This distinction is necessary for editable outputs: an occluded object remains meaningful when another layer is moved or removed.

The offline analysis in Fig. 19 measures endpoint drift inside and outside final alpha support on 18 Full trajectories covering six contents and output counts of four, six, and eight. Each compared region contains at least 20 tokens. At a ten-step window, 72 of 79 eligible layers have lower drift outside support. Averaging within trajectories and then contents gives mean errors of 0.05914 outside and 0.09027 inside support; the mean content-wise relative reduction is 34.32% (95% content-bootstrap interval: 28.17–40.20%). The final alpha map is used here as an offline explanatory variable; online scheduling uses the predicted support.

The decoded replay additionally replaces an equal number of final-latent positions with earlier endpoints, comparing support-outside positions against random positions in the same layer. It measures the local decoded sensitivity of the substituted endpoint. It does not include subsequent

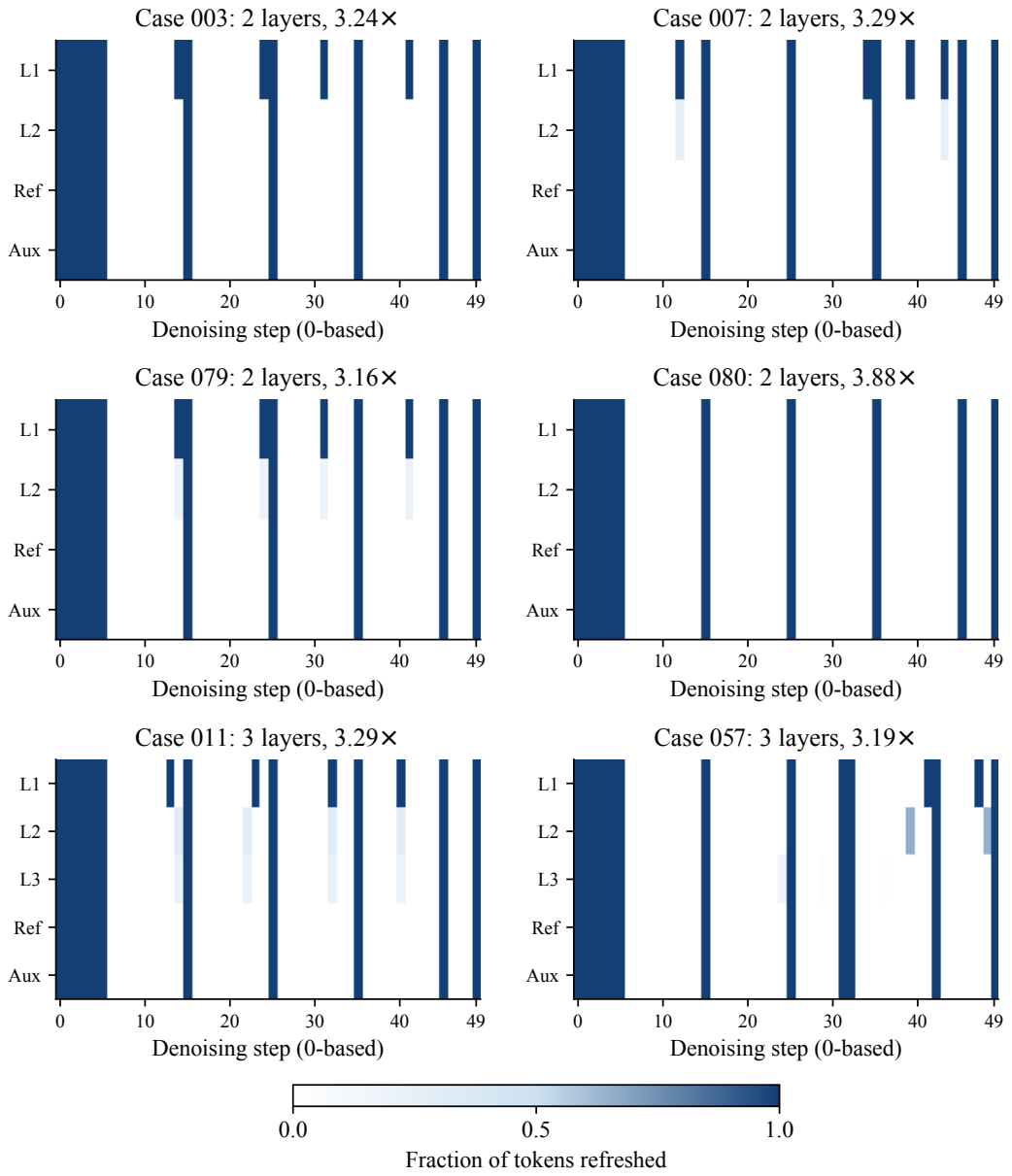

Figure 15: **Actual refresh trajectories, examples 1–6.** These are recorded $\tau = 0.16$ executions, with color proportional to the fraction of each layer's tokens freshly evaluated. Case identifiers match Appendix F.

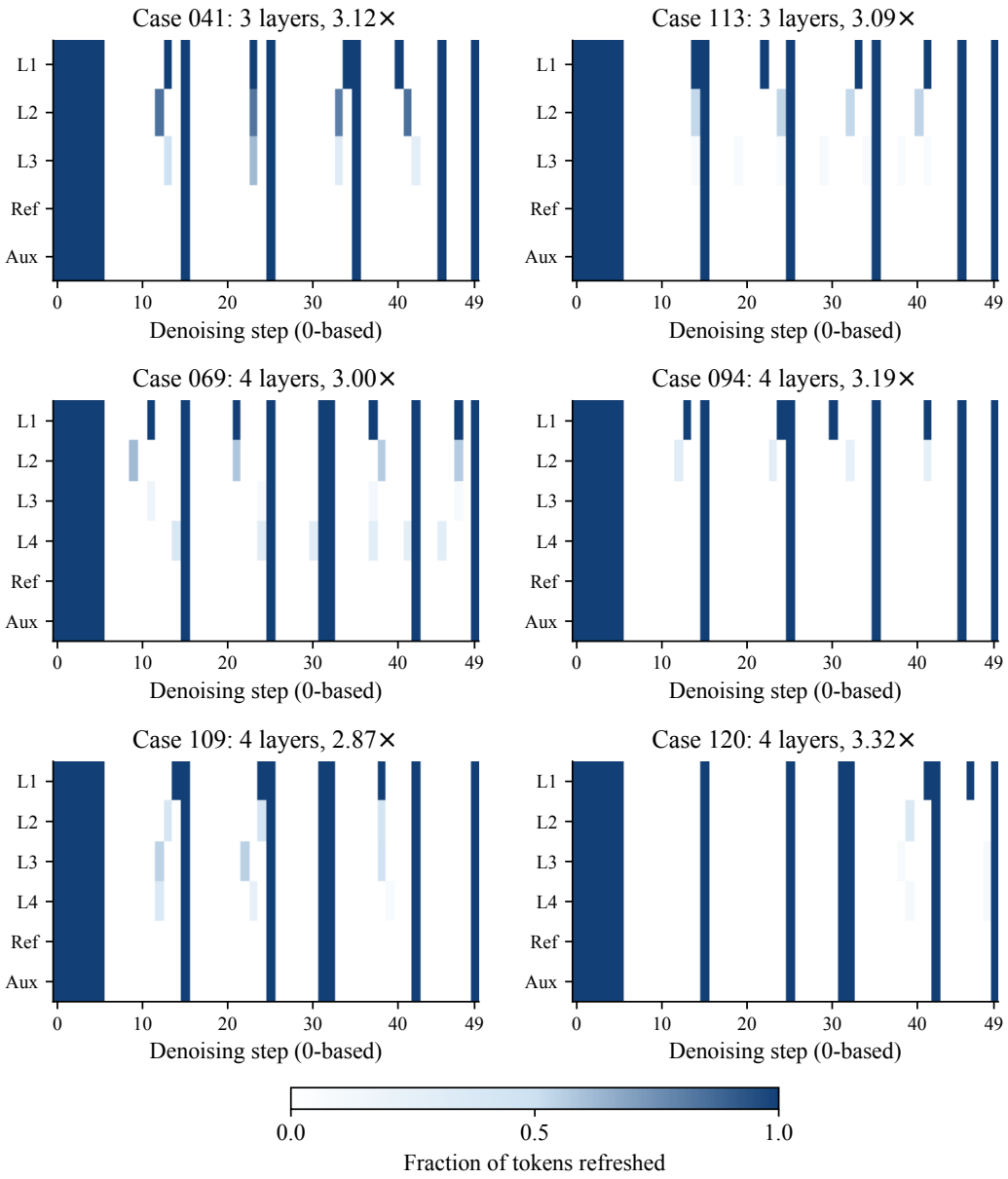

Figure 16: **Actual refresh trajectories, examples 7–12.** Whole-layer reuse and within-layer token restriction coexist in the same trajectory. The color scale and timing convention match Fig. 15.

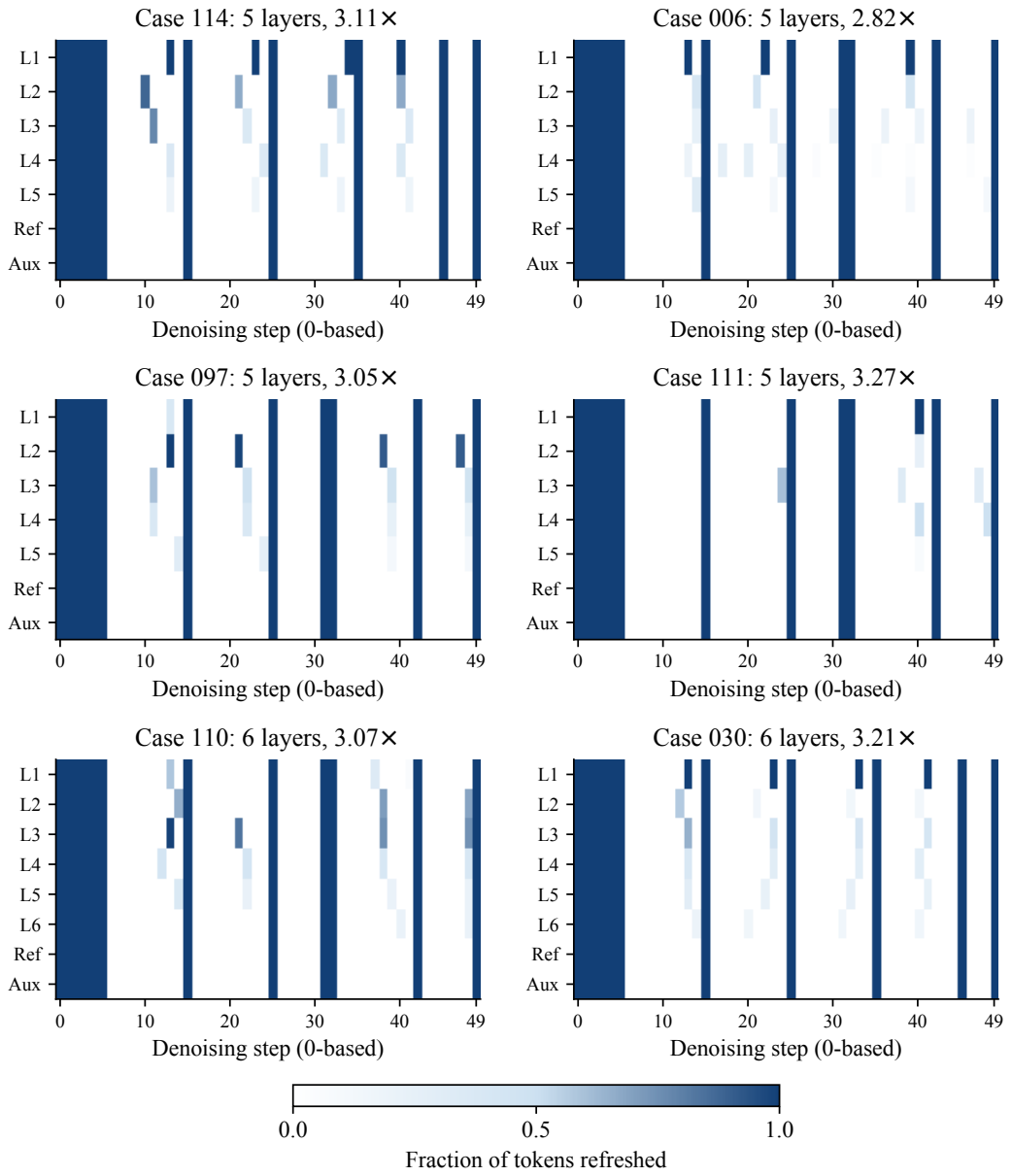

Figure 17: **Actual refresh trajectories, examples 13–18.** Higher-layer-count examples expose distinct refresh patterns across output slots. Ref and Aux rows indicate their recorded refresh events, not additional editable layers.

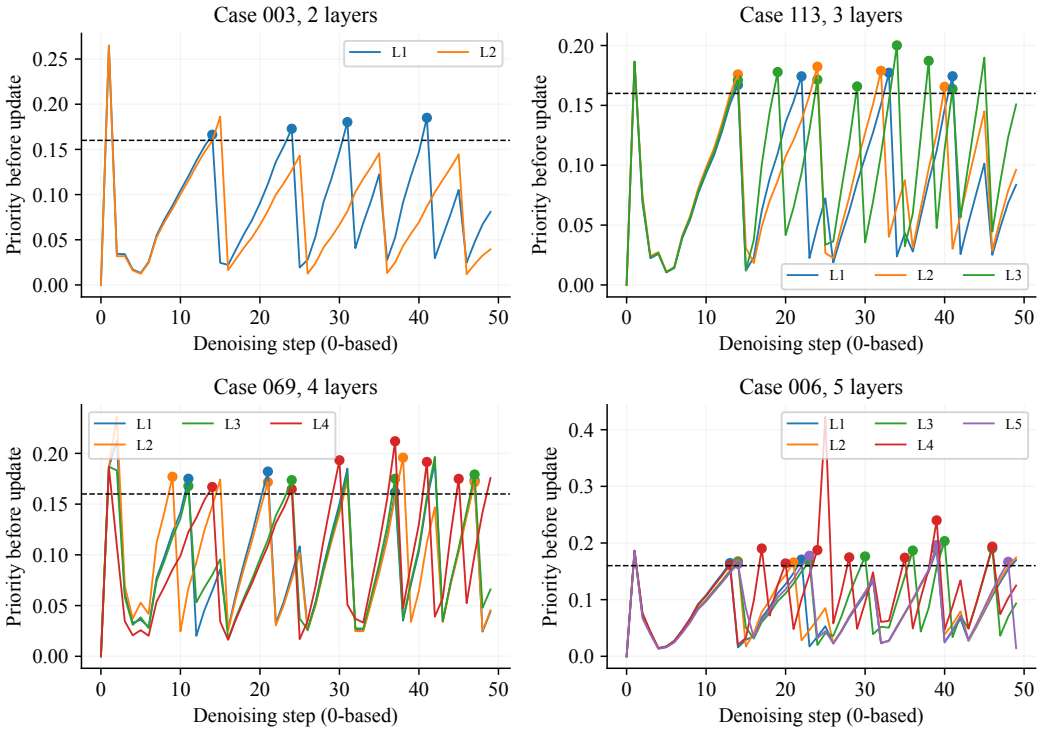

Figure 18: **Online priorities and selective updates.** Lines show recorded pre-update layer priorities; dots identify admission on non-full steps. The dashed line is $\tau = 0.16$. Examples are selected for partial-update visibility from the fixed gallery, while aggregate frequencies use all 100 layouts.

attention feedback from a closed-loop cached trajectory. The component experiment below evaluates the spatial mechanism within actual sampling.

D.5 Temporal, Spatial, and Periodic-Refresh Components

The component study uses 500 layouts selected proportionally by layer-count strata from the fixed cohort, preserving manifest order within each stratum. Temporal-only keeps support-domain scoring and calibration but evaluates all tokens of selected layers. Spatial-only computes the same scores but admits every output layer, leaving only the spatial token restriction. The no-periodic variant disables the ten-step full-refresh interval. All variants retain warmup, final refinement, Ref/Aux timers, endpoint reuse, and exception guards.

Table 20 supplements the main ablation table with indexed-layer measurements. Figure 20 shows all paired contents rather than only aggregate means. Temporal selection accounts for most of the measured runtime reduction in this setting. Adding spatial selection changes mean sampling time from 31.74 to 30.49 seconds, with composite LPIPS of 0.0071 and 0.0072, respectively. The paired complete-minus-temporal RGBA RMSE difference is 0.00438, with a 95% interval of $[-0.00039, 0.01365]$.

Table 20: Layer-level quality in the matched 500-layout component study. Indexed alpha IoU averages corresponding layers; ground-truth metrics use the official edit-aware evaluator. All variants retain warmup and final full updates.

Configuration	Speedup↑	RGBA abs.↓	RGBA rel.↓	Layer αIoU↑	GT αIoU↑	GT RGB L_1 ↓
Full	1.00	0.0000	0.0000	1.0000	0.9110	0.0557
Temporal only	3.07	0.0128	0.0696	0.9759	0.9088	0.0561
Spatial only	1.55	0.0148	0.0772	0.9735	0.9030	0.0615
No periodic refresh	3.33	0.0414	0.2417	0.9090	0.8855	0.0692
HASTE	3.19	0.0172	0.0777	0.9725	0.9095	0.0562

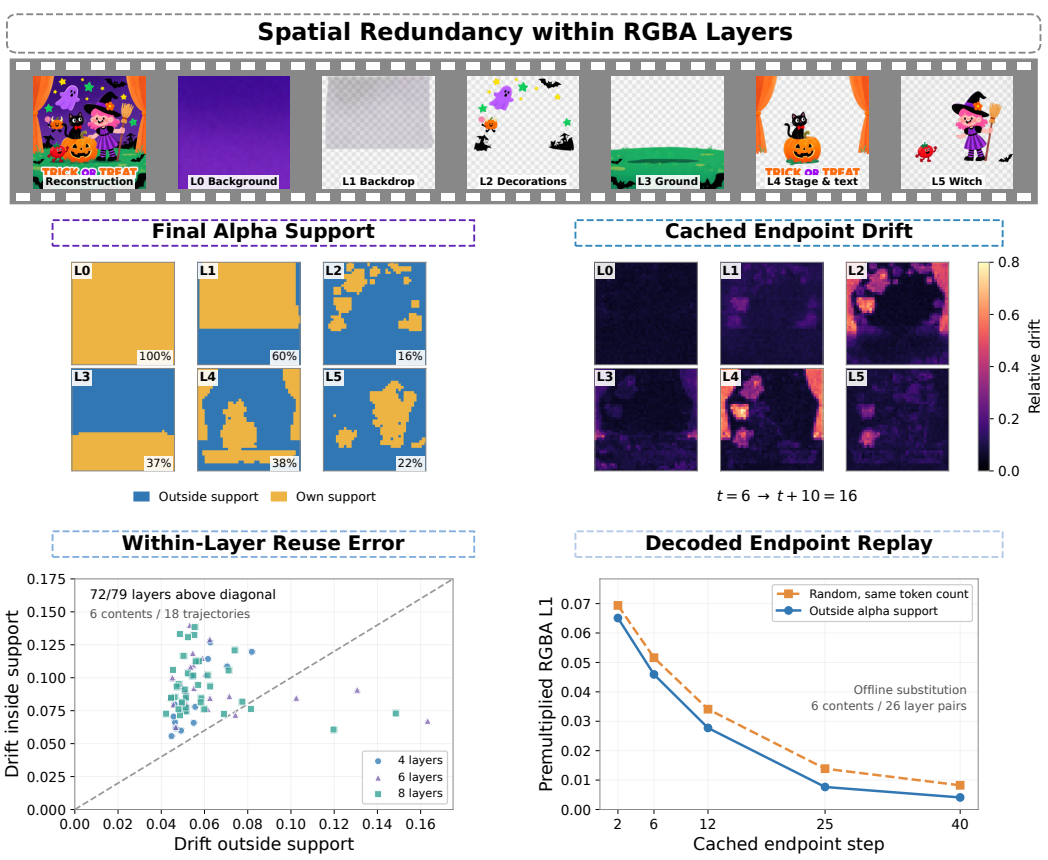

Figure 19: **Spatial support and endpoint reuse.** Top: an illustrative six-layer decomposition, final own-alpha masks, and endpoint drift over the same ten-step window, with a shared drift color scale. Bottom left: all 79 eligible layers from the 18-trajectory analysis. Bottom right: 26 paired layer substitutions at each stored step, comparing outside-support reuse with equal-size random reuse. The illustrative content is separate from the six-content drift aggregate.

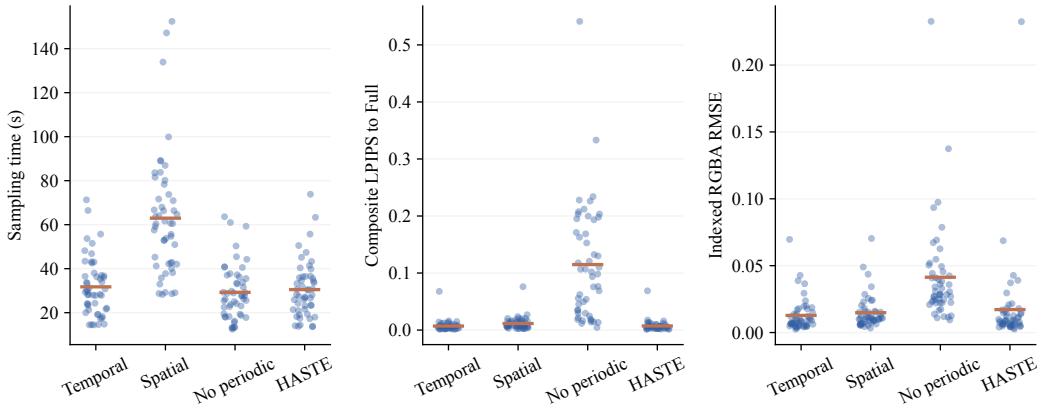

Figure 20: **Per-content component ablations.** Each dot represents one of 50 matched layouts; horizontal orange segments mark means. Temporal and Spatial denote the corresponding single-component variants. All variants use the same paired Full outputs for fidelity evaluation.

Periodic refresh provides support re-exposure and cache renewal. Removing it reduces mean sampling time from 30.49 to 29.25 seconds, while increasing composite LPIPS from 0.0072 to 0.1148 and indexed RGBA RMSE from 0.0172 to 0.0414. This measured trade-off supports retaining periodic refresh in the reported operating point. Selective layer scheduling remains active between these full updates, as shown by the refresh trajectories.

Full and complete-HASTE references reuse the recorded outputs and normal sampling times for these exact identities. The temporal-only and spatial-only runs include execution-audit overhead in their timing. Confidence intervals resample contents 5,000 times and describe variation across layouts, rather than repeated timing batches. No separate FLOPs profile is reported for these two single-component variants.

E Generalization Across Models

E.1 Model-specific execution and shared scheduling

The transfer study preserves each model's native Full sampler and output representation. HASTE applies the same admission threshold 0.16 and EMA new-feedback weight 0.2, while the execution adapter respects the model's conditioning and decoder. Table 21 lists the sampler and refresh settings. CLD, RevealLayer, and Ming use the same 1,000 Crello identities; 100 of these identities participated in target-model refresh configuration selection. Stable-Layers uses the separate 300-layout cohort.

Table 21: Native samplers and transfer configurations. W/F/R denotes warmup steps, final refinement steps, and the periodic full-refresh interval. The table counts sampler steps; Stable-Layers also requires a corrector evaluation.

Model	Layouts	Steps	CFG	W/F/R	Sampling placement
Qwen	1,000	50	4	6/1/10	Two A6000, one CFG branch each
Stable-Layers	300	50	1	6/1/10	A6000, native Heun execution
CLD	1,000	28	3.5	6/1/10	One A6000 for sampling
RevealLayer	1,000	30	3.5	6/1/10	One A6000 for sampling
Ming	1,000	12	2	6/0/10	One A100 per image

Stable-Layers. The Qwen-based LoRA uses a 640-pixel long side, CFG 1, and 50 Heun macro-steps, corresponding to 99 network evaluations. The predictor and corrector share the layer-admission decision. Its $4.26\times$ speedup is measured against its own paired Full sampler on 300 layouts.

CLD. CLD uses its FLUX backbone, 28 Euler steps, true CFG 3.5, guidance embedding 4, and adapter scale 0.9. Sampling, caches, the dense adapter, and native transparent decoder remain on one A6000. Text encoding uses the second GPU before sampling. Alpha estimation follows the native transparent decoder rather than assuming that Qwen's packed latent channels are interchangeable with FLUX features.

RevealLayer. RevealLayer uses 30 steps, a 1024-pixel long side, true CFG 3.5, and guidance embedding 1. Its native Refiner is retained. Backbone K/V caches use FP8, while Refiner caches use BF16. As in CLD, sampling runs on one A6000 and text encoding is prepared on the second GPU.

Both CLD and RevealLayer receive bounding boxes derived from fixed ground-truth groups, shared between Full and HASTE. Their native ROI cropping is part of the Full baseline and is not counted as a caching saving. These conditioned models are compared to their own Full outputs; their absolute task metrics are not ranked against models without these conditions.

Ming-Image-0.1-Design-Layer. Ming uses BF16, its native 12-step schedule, CFG 2, and a 1024-pixel resolution bucket on one A100-SXM4-40GB per image. HASTE uses six warmup steps and no final refinement. With this short trajectory, the recorded runs have no intervening periodic full update. The configuration is evaluated over all 1,000 layouts, giving mean sampling times of 103.71 seconds for Full and 68.30 seconds for HASTE, or $1.52\times$ acceleration. Ground-truth layers are resized to the native output canvas for evaluation.

Ming's support adapter reuses the frozen Qwen AlphaNet after converting the latent scaling, normalization, and packing order. The two RGBA VAE weight files used by this adapter are byte-identical. The conversion first restores the raw VAE latent, applies Qwen's channelwise normalization, and packs the 2×2 subpositions in AlphaNet's expected channel order. AlphaNet weights are not fitted again on Ming outputs.

E.2 Detailed transfer results

Table 22 separates the three 1000-layout evaluations by output-layer count. Each model has a different sequence organization, sampler, and fixed execution overhead, so transfer is assessed by paired acceleration and fidelity within that model.

Table 22: Transfer results grouped by output-layer count, using the same 1000 identities for each model. Each speedup uses its own Full reference. $\Delta\alpha$IoU is **HASTE** minus Full at edit budget five.

Model	N	n	Speedup↑	Time (s)	$\Delta\alpha$IoU	LPIPS↓
CLD	2	290	1.90	32.45	-0.0004	0.0100
CLD	3	340	1.73	45.81	-0.0007	0.0137
CLD	4	200	1.64	50.67	+0.0101	0.0302
CLD	5	150	1.59	59.94	+0.0003	0.0449
CLD	6	20	1.33	94.68	+0.0007	0.0774
RevealLayer	2	290	2.22	72.80	+0.0005	0.0225
RevealLayer	3	340	2.11	89.24	-0.0012	0.0290
RevealLayer	4	200	1.81	146.90	-0.0111	0.0496
RevealLayer	5	150	1.68	198.28	+0.0033	0.0311
RevealLayer	6	20	1.67	198.06	+0.0013	0.0524
Ming	2	290	1.59	41.96	-0.0005	0.0079
Ming	3	340	1.50	61.72	-0.0023	0.0095
Ming	4	200	1.52	84.23	-0.0021	0.0152
Ming	5	150	1.50	105.50	-0.0019	0.0191
Ming	6	20	1.46	123.94	-0.0013	0.0150

F Additional Qualitative Comparisons

F.1 Selection and presentation

The fixed Qwen gallery contains 18 layouts spanning two to six output layers. Within each layer count, identities are sorted lexicographically and four evenly spaced identities are selected, or all identities when fewer than four are available. Selection is independent of quality and speedup. All Qwen results use $\tau = 0.16$.

The input/composite overview retains all 18 layouts. Detailed comparisons show a subset of these layouts while preserving every output layer of each shown example. Columns follow the native back-to-front order; checkerboards indicate transparency. Original aspect ratios are retained, with only Qwen inference padding removed. Figure labels report per-example sampling speedups.

F.2 Comparison with caching baselines

Figures 21–25 show one layout at each output-layer count from two to six. We retain the lexicographically first identity at each count in the fixed gallery. Rows compare Full, **HASTE**, PAB, ToCa, and Jano, including the composite and all output layers.

Inputs, conditioning, initial noise, and 50 sampling steps are shared. Baseline settings follow Appendix A.4. Speedups use complete sampling-loop times; ToCa's recorded host-cache transfers are included for cases 069 and 110.

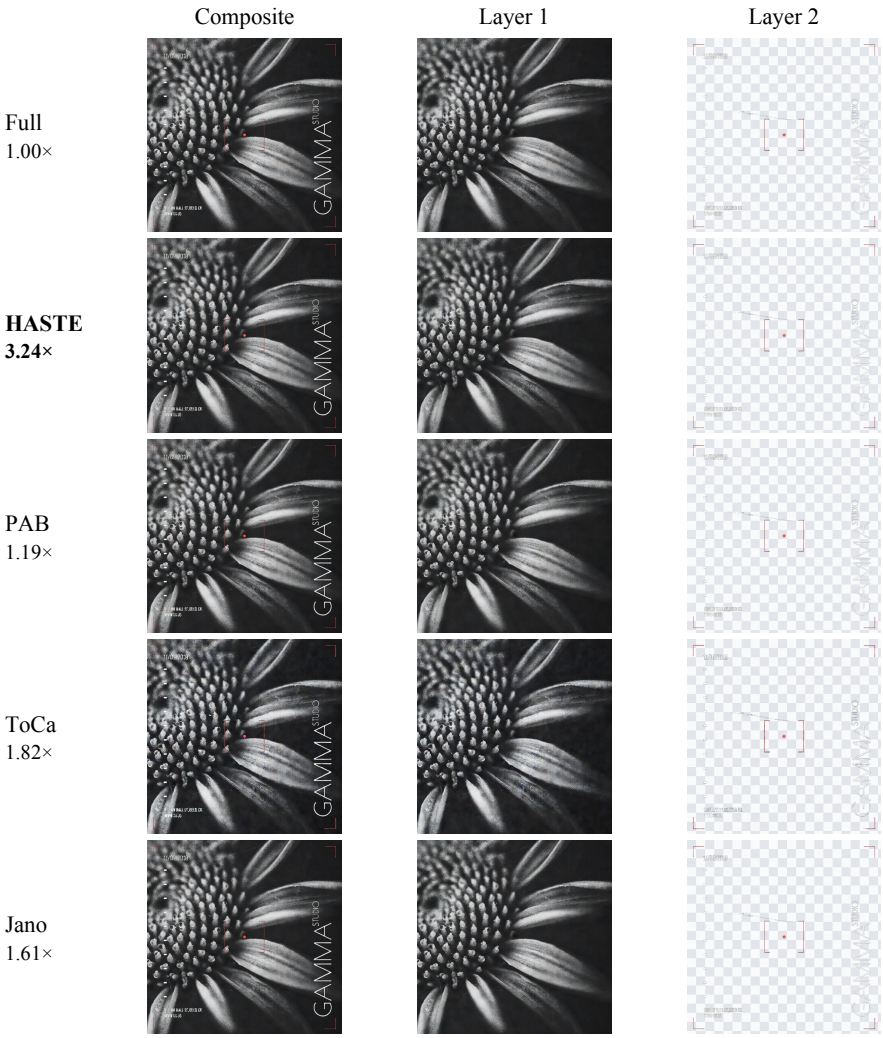

Figure 21: **Case 003, 2 output layers.** Composite and all RGBA layers for Full, **HASTE**, PAB, ToCa, and Jano; labels give per-example speedups.

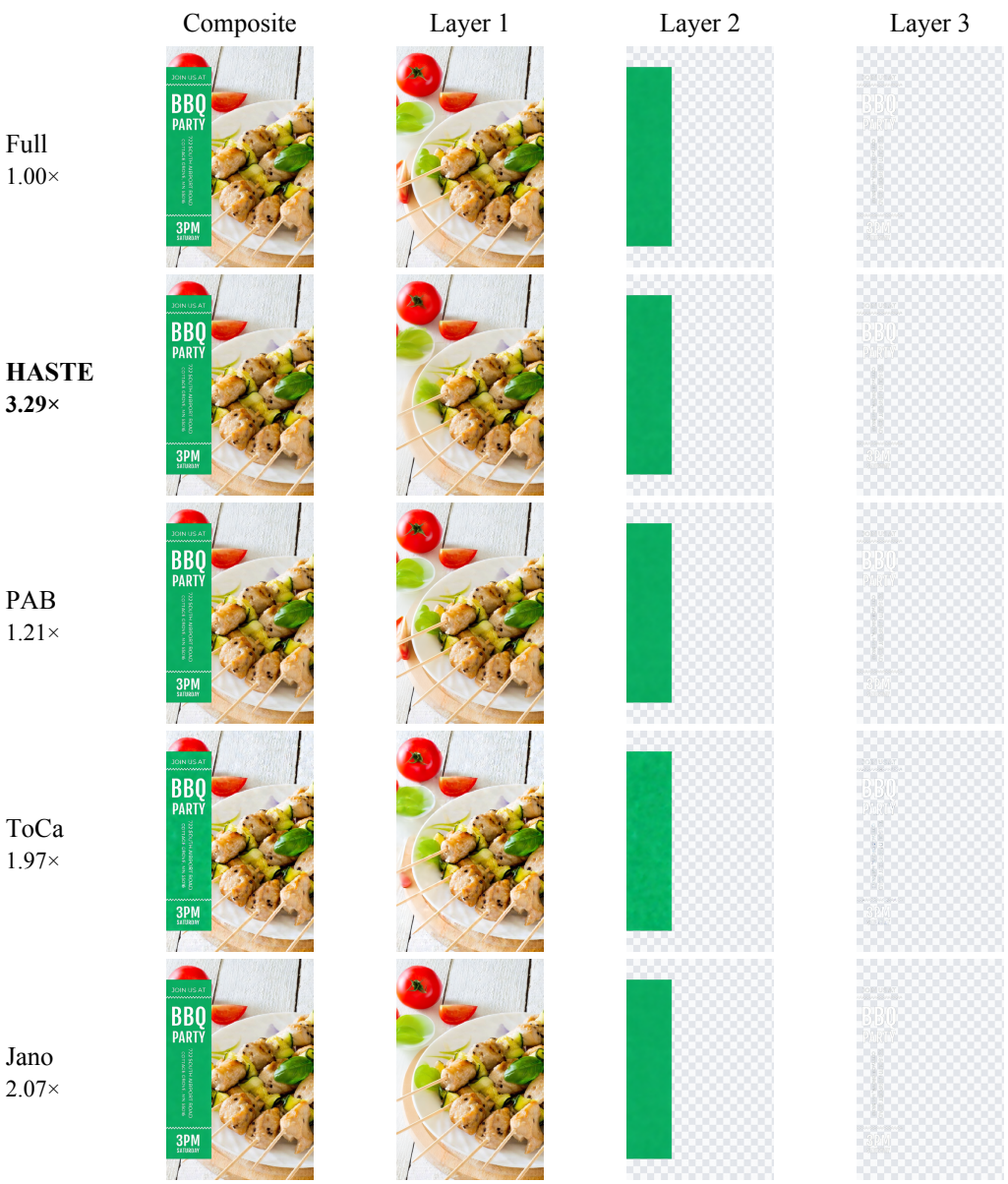

Figure 22: **Case 011, 3 output layers.** Composite and all RGBA layers for Full, **HASTE**, PAB, ToCa, and Jano; labels give per-example speedups.

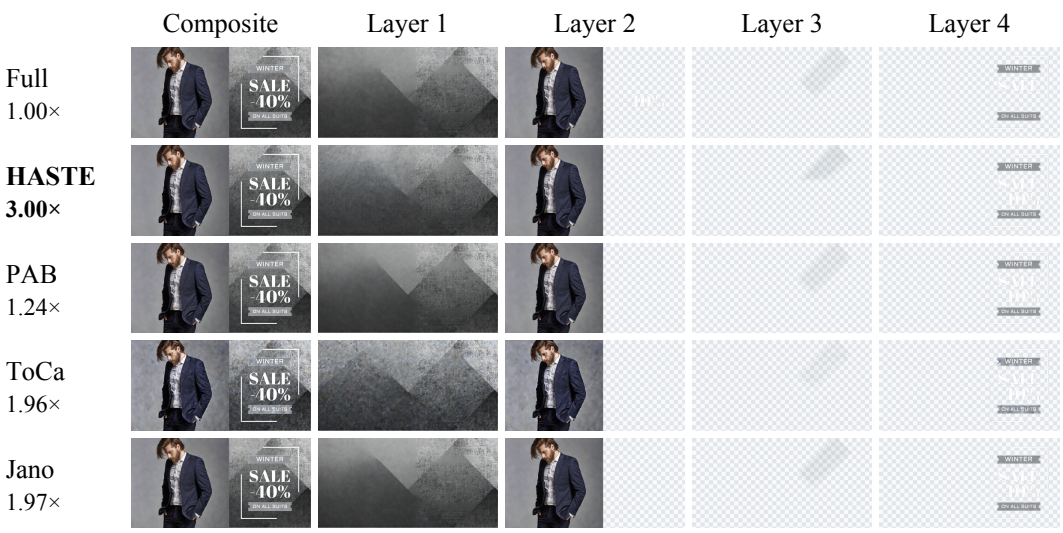

Figure 23: **Case 069, 4 output layers.** Composite and all RGBA layers for Full, **HASTE**, PAB, ToCa, and Jano; labels give per-example speedups. ToCa includes recorded host-cache transfers.

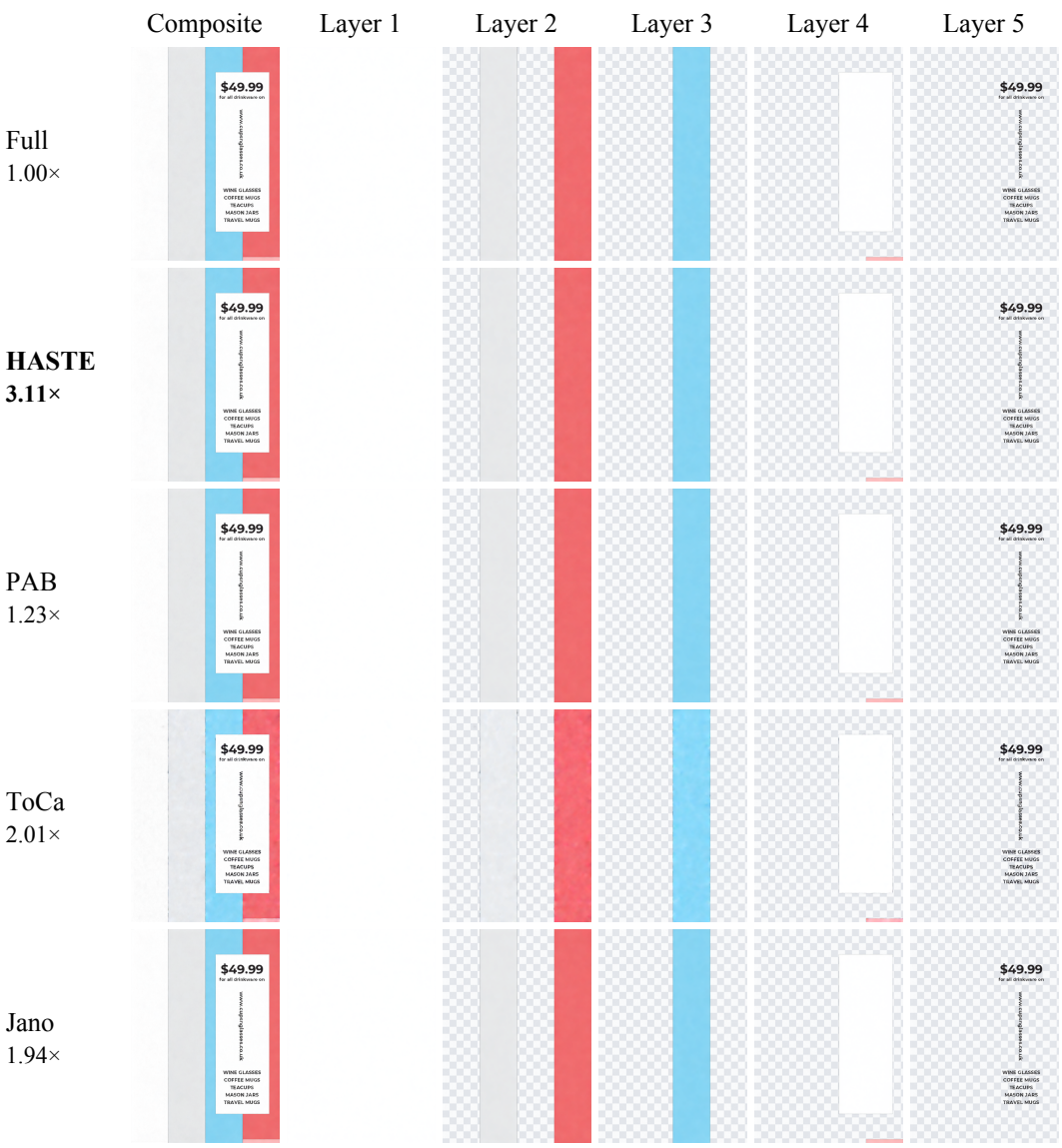

Figure 24: **Case 114, 5 output layers.** Composite and all RGBA layers for Full, HASTE, PAB, ToCa, and Jano; labels give per-example speedups.

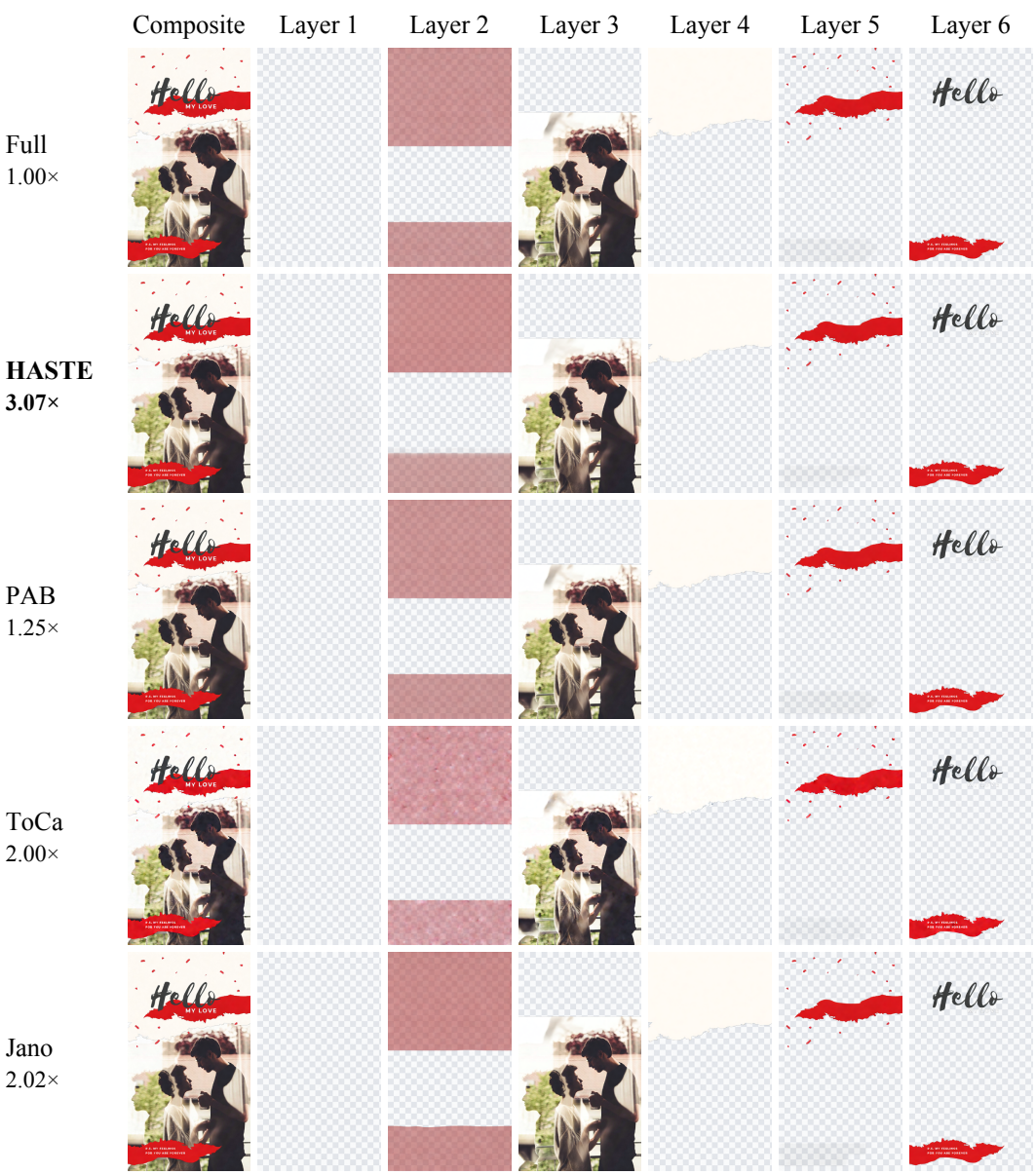

Figure 25: **Case 110, 6 output layers.** Composite and all RGBA layers for Full, HASTE, PAB, ToCa, and Jano; labels give per-example speedups. ToCa includes recorded host-cache transfers.

F.3 PAIRED FULL/HASTE DECOMPOSITIONS

Figures 26 and 27 summarize inputs and composites for the complete 18-layout gallery. The detailed plates in Figs. 28, 29, and 30 retain examples 4–6, 10–12, and 16–18 in the fixed gallery order. These nine layouts cover all five output-layer counts, with every generated slot shown.

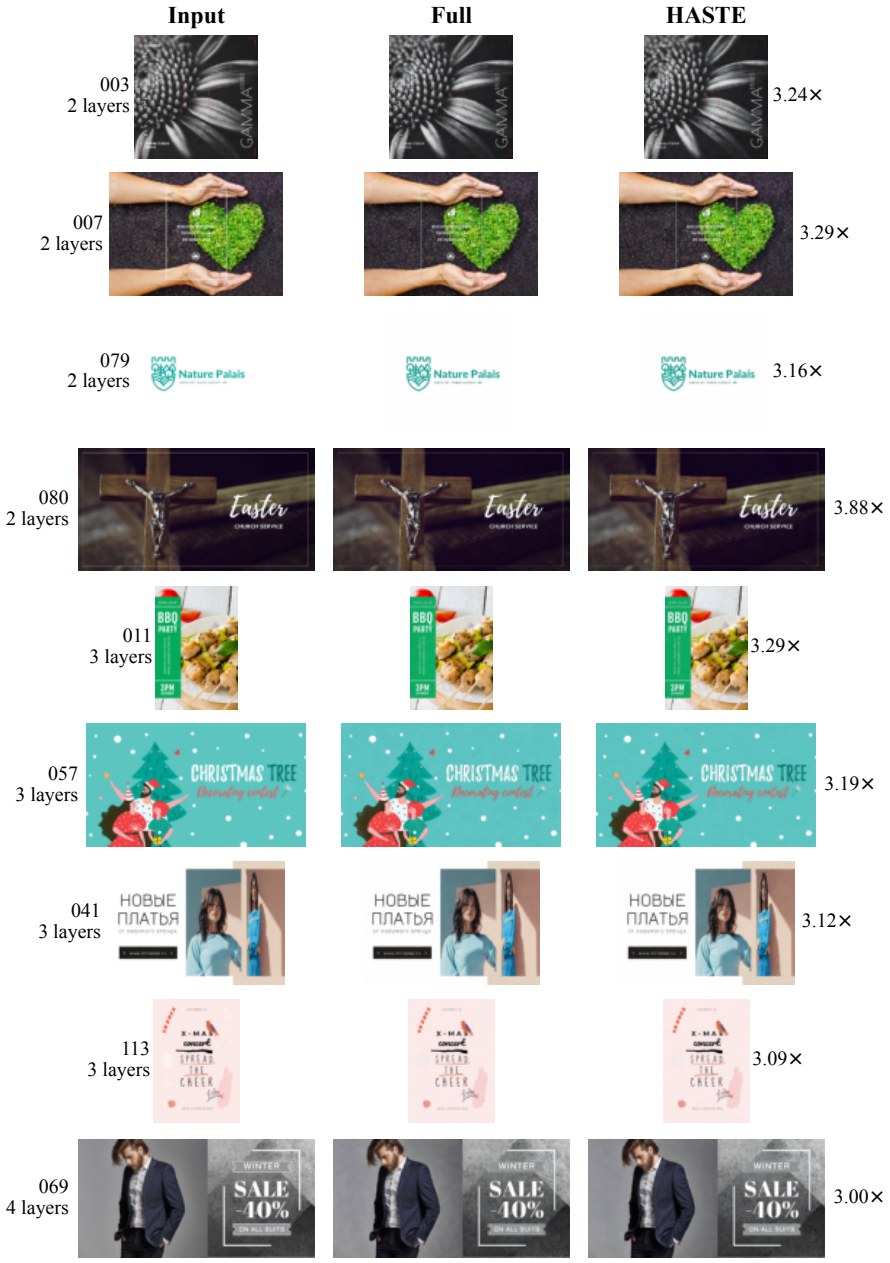

Figure 26: **Input and composite comparisons, examples 1–9.** The fixed gallery selection spans different aspect ratios, semantic contents, and layer counts. Full and HASTE share conditioning, initial noise, and 50 sampling steps.

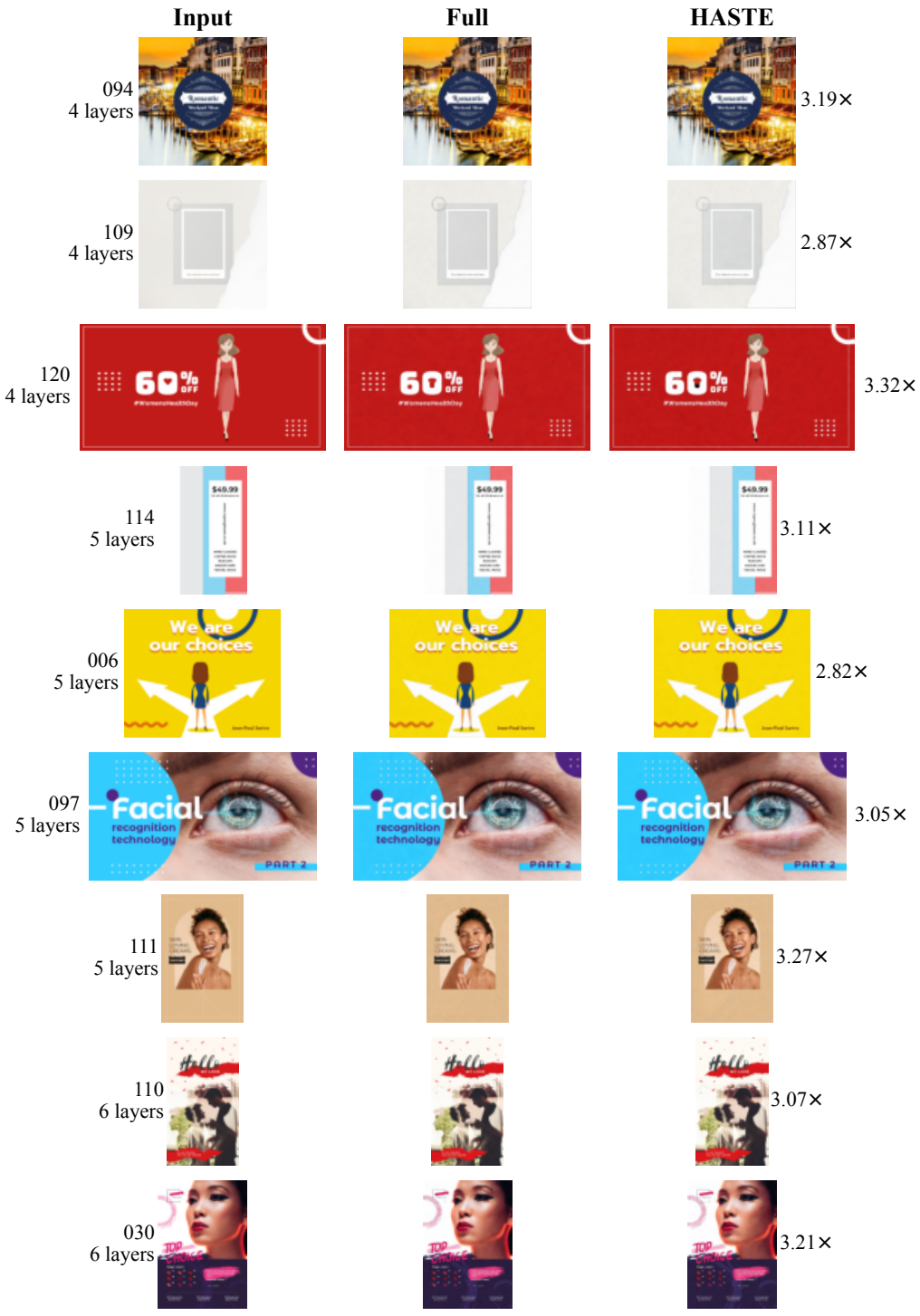

Figure 27: **Input and composite comparisons, examples 10–18.** The presentation and selection rule match Fig. 26. Corresponding complete layer decompositions follow below.

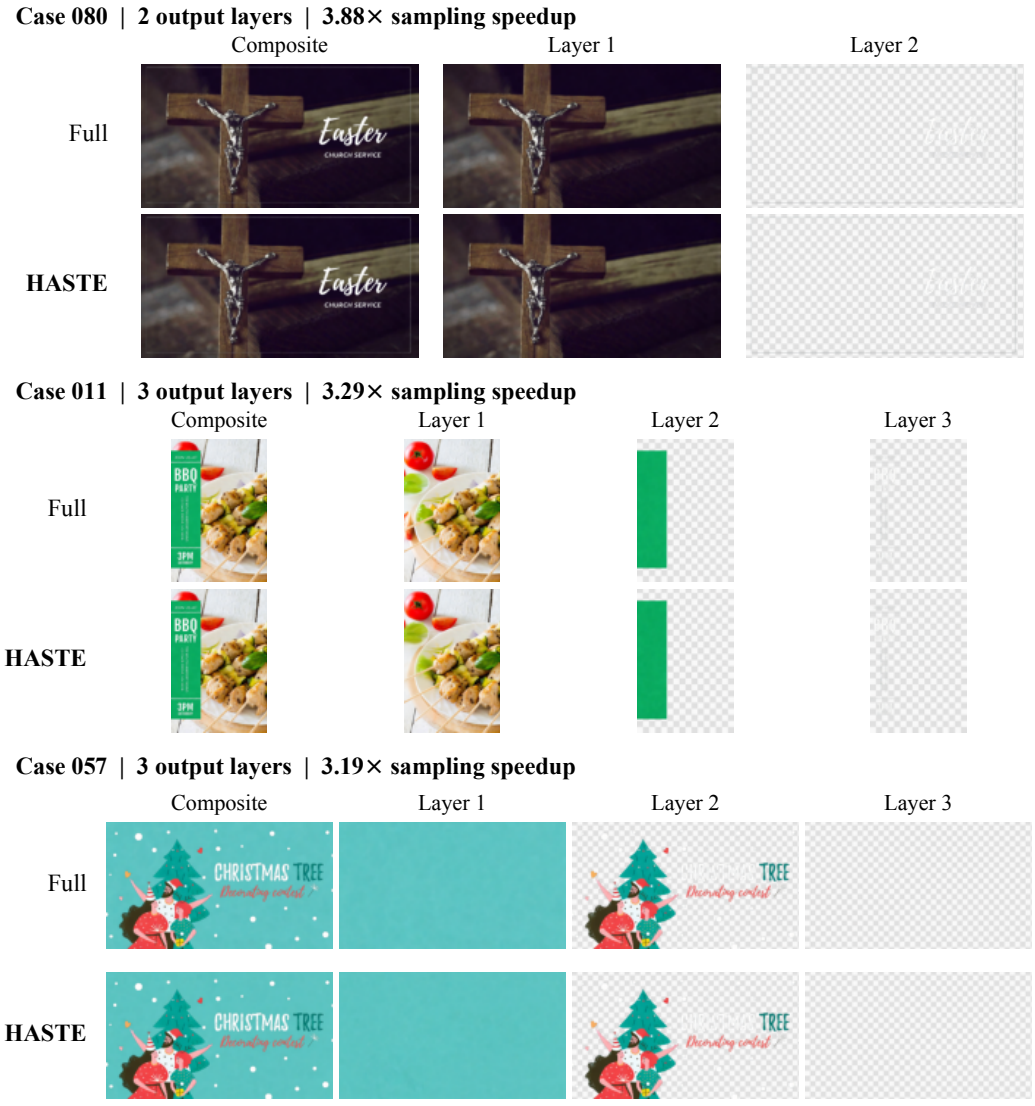

Figure 28: **Complete Qwen decompositions, examples 4–6.** Each Full/HASTE pair uses identical inputs and initial noise. Transparent regions are visualized on the same checkerboard.

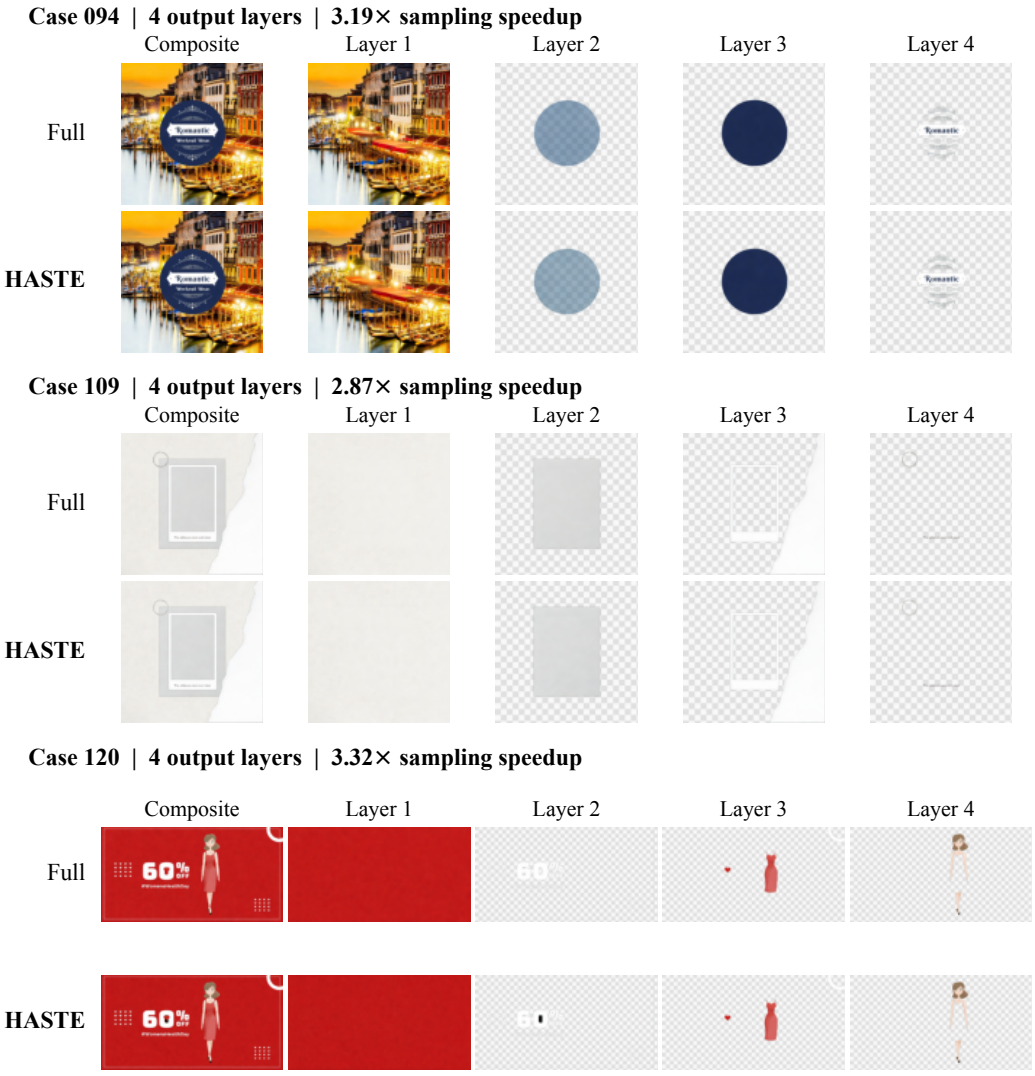

Figure 29: **Complete Qwen decompositions, examples 10–12.** The four-layer cases use the same admission threshold and refresh settings as the lower-layer-count cases.



Figure 30: **Complete Qwen decompositions, examples 16–18.** This plate includes both six-layer layouts in the fixed 100-layout cohort. No layer is hidden or reassigned for visualization.

F.4 CROSS-MODEL PAIRED DECOMPOSITIONS

Figures 31–33 retain the five- and six-layer examples for CLD, RevealLayer, and Ming. The same two identities are used for all three models, selected as the lexicographically first identity at each layer count. Each Full/HASTE pair preserves native conditioning and output order. CLD and RevealLayer share their group-derived bounding boxes across methods; Ming uses its native input conditioning. All generated slots are shown, complementing the aggregate results in Appendix E.

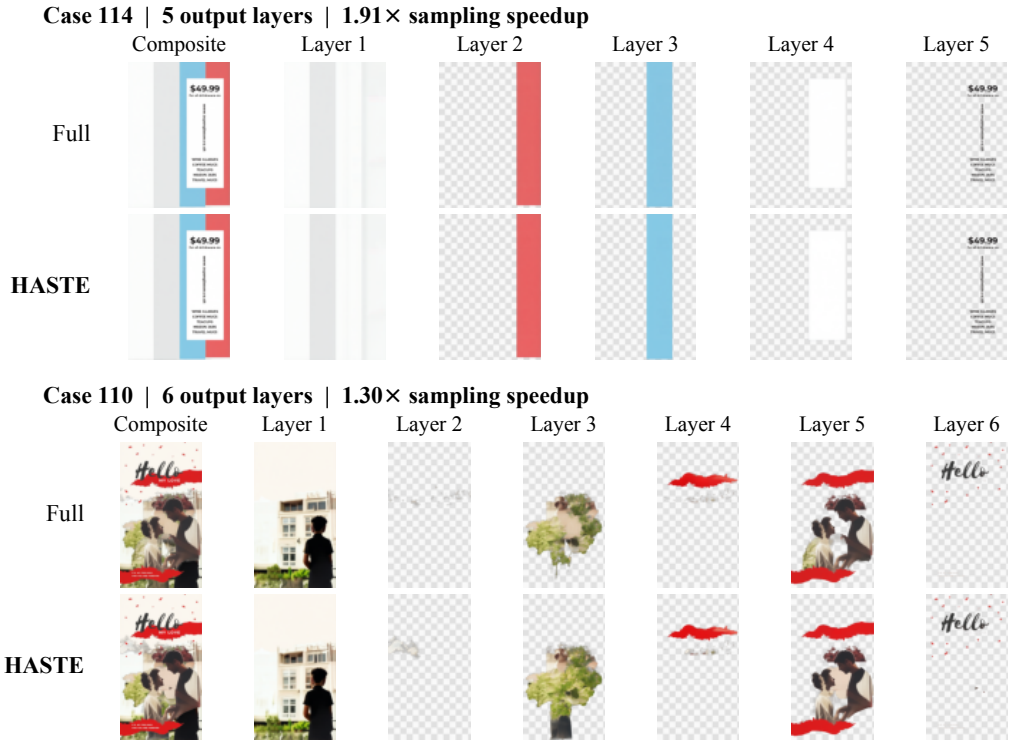

Figure 31: **CLD transfer: five- and six-layer examples.** All output slots are shown. Displayed speedups compare paired sampling times on the same A6000 execution layout.

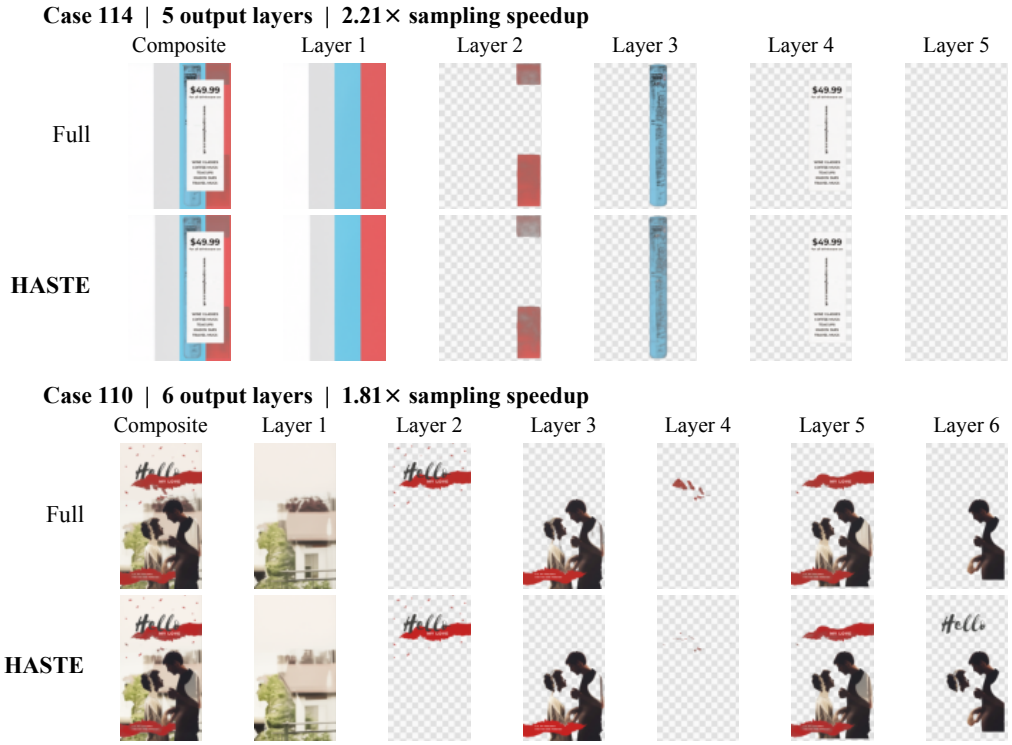

Figure 32: **RevealLayer transfer: five- and six-layer examples.** The selected identities match the CLD and Ming plates; layer groupings remain model-specific.

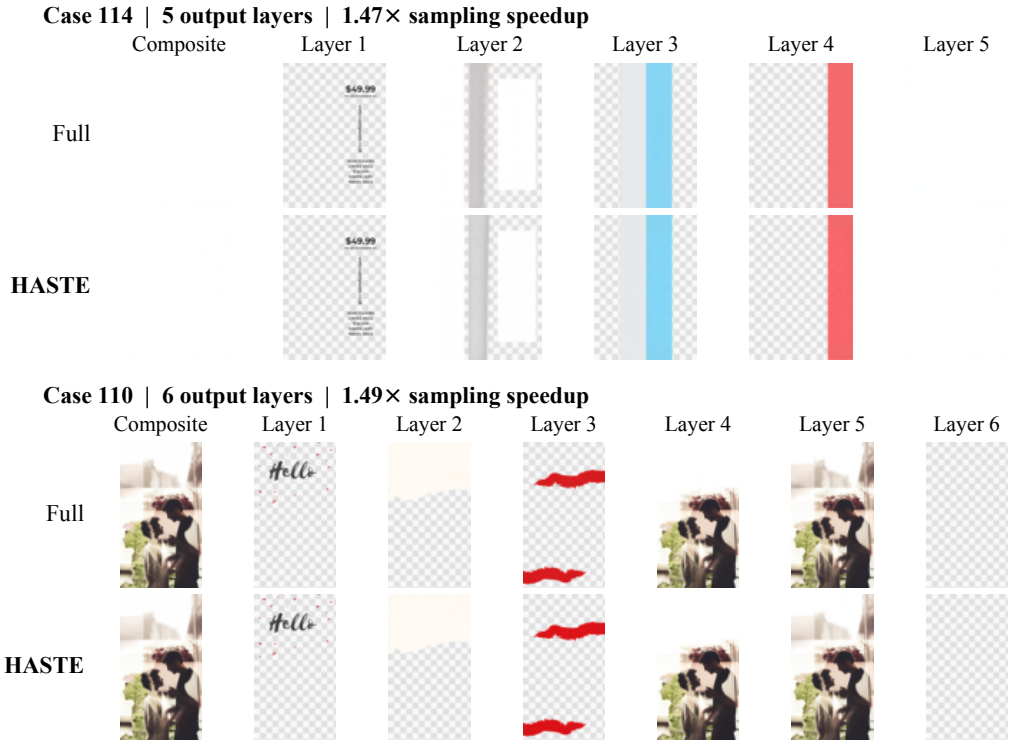

Figure 33: **Ming transfer: five- and six-layer examples.** All layers and the recomposed image are shown at the native output aspect ratio. Sampling conditions match the 100-layout Ming evaluation.