# ToMA: Token Merge with Attention for Diffusion Models

**Wenbo Lu** [*1] **Shaoyi Zheng** [*1] **Yuxuan Xia** [1] **Shengjie Wang** [1]

## Abstract

Diffusion models excel in high-fidelity image generation but face scalability limits due to transformers' quadratic attention complexity. Plug-and-play token reduction methods like ToMeSD and ToFu reduce FLOPs by merging redundant tokens in generated images but rely on GPU-inefficient operations (e.g., sorting, scattered writes), introducing overheads that negate theoretical speedups when paired with optimized attention implementations (e.g., FlashAttention). To bridge this gap, we propose **To**ken **M**erge with **A**ttention (ToMA), an off-the-shelf method that redesigns token reduction for GPU-aligned efficiency, with three key contributions: 1) a reformulation of token merge as a submodular optimization problem to select diverse tokens; 2) merge/unmerge as an attention-like linear transformation via GPU-friendly matrix operations; and 3) exploiting latent locality and sequential redundancy (pattern reuse) to minimize overhead. ToMA reduces SDXL/Flux generation latency by 24%/23% (with DINO $\Delta < 0.07$), outperforming prior methods. This work bridges the gap between theoretical and practical efficiency for transformers in diffusion.

## 1. Introduction

Diffusion models (Ho et al., 2020; Song et al., 2021; Dhariwal & Nichol, 2021) have revolutionized high-fidelity image generation. Yet, their reliance on transformer architectures—notably in U-ViT (Bao et al., 2023) and DiT (Peebles & Xie, 2023)—introduces a bottleneck: the quadratic complexity of self-attention scales prohibitively with token counts, exacerbating latency across denoising steps.

Efforts to accelerate transformers broadly fall into two categories: attention optimization (e.g., Flash Attention (Dao, 2023), xformers (Lefaudeux et al., 2022)) and token reduction. While there is extensive research on token reduction for *discriminative tasks*, including Token Merge (Bolya et al., 2023), AdaViT (Yin et al., 2022b), and later improvements like Diffrate (Chen et al., 2023), these approaches often employ irreversible token pruning or merging, which is unsuitable for *generative tasks*.

Generative architectures like diffusion models impose stricter constraints: token counts must be restored after merging (i.e., "unmerging") to preserve spatial consistency for iterative refinement. Earlier attempts, including ToMeSD (Bolya & Hoffman, 2023), Token Pruning (Kim et al., 2022), and ToFu (Kim et al., 2023) adapt token reduction to this setting but suffer from a critical flaw: their merge/prune (and unmerge/fill) operations rely on GPU-inefficient primitives (e.g., sorting, scattered memory writes). This problem intensifies when paired with highly optimized attention implementations like those in the `diffusers` framework (von Platen et al., 2022).

In such scenarios, the attention mechanism, once the primary bottleneck, is streamlined to approach hardware efficiency limits, leaving only marginal time reductions attainable. Consequently, these potential gains are dwarfed by the computation costs of unoptimized merging logic, which now dominates the computation time. In other words, the overhead introduced by previous token merging methods becomes negligible only when applied with fast implementations of attention, thereby preventing practical speed-ups.

In this paper, we propose **To**ken **M**erge with **A**ttention (**ToMA**), a training-free framework that bridges this gap by rethinking token merge for GPU-aligned operation. To achieve practical speed-up without image degradation, we reformulate token merge as a submodular optimization problem, leveraging its theoretical guarantees and algorithmic toolkit. Specifically, ToMA uses our GPU-optimized facility location algorithm to select a diverse and representative set of "destination" tokens. Merge is then formulated as an attention-like linear transformation for efficient aggregation of non-destination tokens, with unmerge implemented via its inverse transformation, making full use of hardware-friendly matrix multiplications. This approach down-projects the latent space while preserving critical information, accelerating

[*]Equal contribution [1]Department of Computer Science, New York University. Correspondence to: Wenbo Lu <wenbo.lu@nyu.edu>, Shaoyi Zheng <sz3684@nyu.edu>, Yuxuan Xia <yx2432@nyu.edu>, Shenji Wan <sw5973@nyu.edu>.

*Proceedings of the $42^{nd}$ International Conference on Machine Learning*, Vancouver, Canada. PMLR 267, 2025. 


*Figure 1.* Comparison on `SDXL-base` with four configurations (left to right): Original, +FA2, +ToMeSD, +ToMA (on top of FA2, ratio=0.5). While ToMeSD fails to speed up due to overhead, ToMA achieves significant acceleration with negligible loss in image quality.

transformers with negligible overhead or quality loss. To further minimize computational overhead, we exploit two intrinsic properties of diffusion models:

- Latent Space Locality: Tokens exhibit spatial coherence, allowing parallel merging within non-overlapping local windows (e.g., 8×8 patches).
- Sequential Redundancy: Merge patterns persist across 1) adjacent denoising timesteps; and 2) consecutive transformer layers. We amortize the overhead by reusing merge patterns over multiple steps and layers.

Theoretical guarantees from submodular optimization ensure that ToMA's token selection approximates optimal coverage. At the same time, its co-design with GPU execution paradigms (e.g., batched matrix operations) eliminates costly operations inherent in prior methods. This synergy translates to **real-world speedup**, rather than theoretical FLOP reductions only. For example, ToMA reduces the total generation time for `SDXL-base` by 24% and `Flux.1-dev` by 23% with negligible degradation of image quality (change in DINO score $< 0.07$), outperforming previous work like ToMeSD and ToFu, which either fails to accelerate modern attention implementations or introduce artifacts at comparable compression rates. Our contributions are summarized:

- Algorithmic Innovation: A submodular optimization framework for token merging, ensuring provably representative token selection to enhance quality.
- System Co-Design: GPU-aligned implementation strategies leveraging invertible, attention-like operations to exploit latent space locality and temporal redundancy, minimizing computational overhead.
- Empirical Validation: ToMA achieves at least $1.24\times$ practical speedup when paired with FlashAttention2, State-of-the-art results across different diffusion models (e.g., `SDXL-base`, `Flux.1-dev`) and GPU architectures (NVIDIA RTX6000, V100, RTX8000).

## 2. Related Work

**Efficient Vision Transformer** Vision Transformers (Dosovitskiy et al., 2021) face computational challenges due to quadratic attention complexity. Recent efforts to mitigate this fall into four categories: compact architectures (e.g., Swin Transformer (Liu et al., 2021a), PVT (Wang et al., 2021)), pruning strategies like X-Pruner (Yu & Xiang, 2023), knowledge distillation such as DeiT (Touvron et al., 2021), and post-training quantization (Liu et al., 2021b). Complementary efforts also explore combined techniques (Papa et al., 2024) to unify these paradigms. While effective, most solutions require retraining and remain inherently token-centric. In contrast, ToMA introduces training-free token merging, operating orthogonally, thus enabling seamless integration without conflict.

**Learned Token Reduction** Most learned token reduction involves training auxiliary models to assess the importance of tokens in the input data. DynamicViT (Rao et al., 2021) employs a lightweight MLP to generate pruning masks based on input token features. A-ViT (Yin et al., 2022a) computes halting probabilities using specific channels of token features to determine the necessity of further processing. Whereas these methods often require additional training for the auxiliary modules, our approach is directly applicable, offering a more generalizable solution.

**Heuristic Token Reduction** Heuristic token reduction strategies can be applied to ViTs without additional training. Our method also falls into this category, making the approaches below natural baselines for comparison. Adaptive Token Sampling (ATS) (Fayyaz et al., 2022) keeps tokens most similar to the class token, which limits its use in pixel-level generation tasks where a class token is absent. Token Downsampling (ToDo) (Smith et al., 2024) downsamples only the key–value in attention, skipping queries, which limits acceleration and causes fine-grained detail loss due to uniform spatial pooling. Token Merging for Stable Diffusion (ToMeSD) (Bolya & Hoffman, 2023) forms

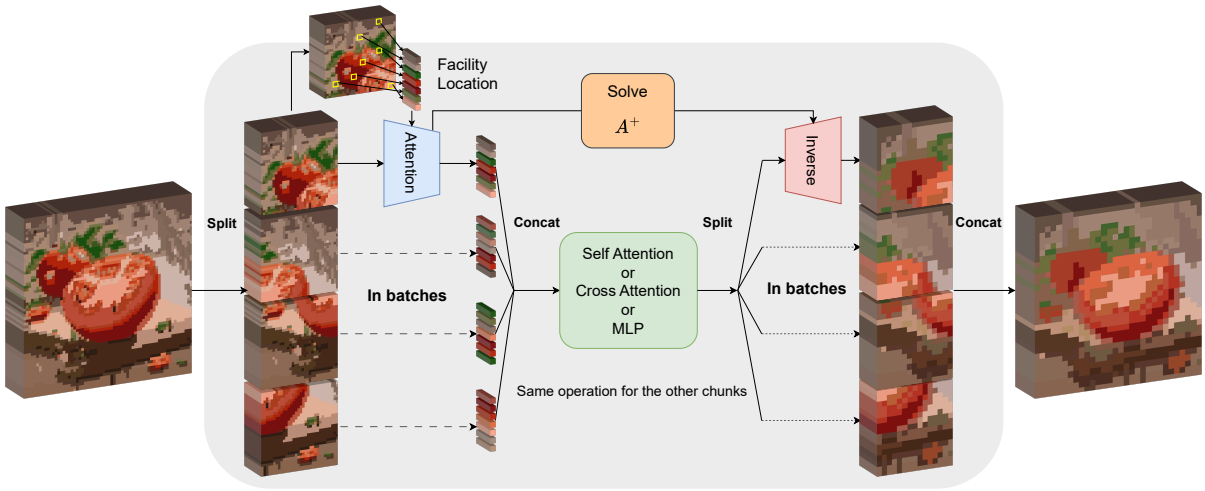


*Figure 2.* Architectural overview of ToMA. The framework consists of three key stages: (1) **Facility Location Algorithm** identifies the best representative token set $D \subset N$ through submodular optimization to maximize representational diversity; (2) **Attention (Merge)** constructs an efficient low-rank attention matrix that maps $N \to D$ via a linear transformation for transformer computation (SelfAttn, CrossAttn, MLP) in the reduced space; (3) **Inverse (Unmerge)** applies the pseudo-inverse to recover full-resolution features $D \to N$. The pipeline operates through localized processing of latent space regions with parallel batch optimization for efficiency.

source–destination pairs within fixed or randomly tiled regions and greedily matches them based on similarity, performing unweighted merging followed by a simple unmerge that copies the destination embedding back to each source position. ToFu (Kim et al., 2023) builds on ToMeSD by dynamically deciding, for each layer, whether to merge or prune tokens according to a linearity test, thereby combining the benefits of both operations.

Prior token-reduction methods share two flaws: GPU-inefficient matching/merging operations and heuristic designs lacking theoretical guarantees of information preservation. By contrast, ToMA employs a GPU-friendly attention-like linear projection whose destinations are chosen via a submodular objective, providing both hardware efficiency and a principled foundation. Moreover, ToMA remains compatible with orthogonal blended schemes (alternating between pruning and merging) such as ToFu, allowing additional speed–quality trade-offs when combined.

## 3. Preliminaries

### 3.1. Attention Notation

The standard Scaled Dot-Product Attention (SDPA) mechanism is widely adopted in mainstream diffusion models. For clarity, we define the following notations: $B$ for batch size, $N$ for sequence length, $d$ for feature dimension, $D$ for the number of all destination tokens, $\mathbf{X} \in \mathbb{R}^{B\times N\times d}$ for the attention input latent tensor, and $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{B\times N\times d}$ for query, key, and value tensors, respectively, projected from $\mathbf{X}$.

The SDPA operation is defined as

$$\text{SDPA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}, \tau) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\tau\sqrt{d}}\right)\mathbf{V}. \quad (1)$$

### 3.2. Submodularity

A submodular function (Fujishige, 2005) is a set function $f : 2^{\mathbb{V}} \to \mathbb{R}$ defined over subsets of the ground set $\mathbb{V}$. It satisfies the diminishing returns property, which states that the marginal gain of adding a new element $v$ to the set decreases as the context set grows. Mathematically:

For any subsets $\mathbb{A} \subseteq \mathbb{B} \subseteq \mathbb{V}$ and element $v \in \mathbb{V} \setminus \mathbb{B}$:

$$f(v|\mathbb{A}) \geq f(v|\mathbb{B}),$$

where the marginal gain $f(\cdot|\cdot)$ is defined as:

$$f(v|\mathbb{A}) \equiv f(\mathbb{A} \cup \{v\}) - f(\mathbb{A}).$$

**Algorithm 1:** Greedy Algorithm

**Input:** Ground set $\mathbb{V}$, submodular function $f : 2^{\mathbb{V}} \to \mathbb{R}$, and budget $k$
**Output:** Selected subset $\mathbb{A}$ of size at most $k$
Initialize $\mathbb{A} \leftarrow \emptyset$;
**for** $i = 1$ *to* $k$ **do**
  Select $v^* = \arg\max_{v' \in \mathbb{V}\setminus\mathbb{A}} f(v'|\mathbb{A})$;
  Update $\mathbb{A} \leftarrow \mathbb{A} \cup \{v^*\}$;
**return** $\mathbb{A}$;

This property makes submodular functions well-suited for modeling diversity and coverage in subset selection problems. Naturally, this leads to the canonical problem of submodular maximization under a cardinality constraint:

$$\max_{\mathbb{A}\subseteq\mathbb{V}} f(\mathbb{A}) \quad \text{s.t. } |\mathbb{A}| \leq k.$$

An intuitive approach is the greedy algorithm (Alg. 1), which guarantees a $(1-1/e)$-approximation of the optimal solution (Nemhauser et al., 1978). Starting with $\mathbb{A}=\emptyset$, the algorithm iteratively selects the element with the highest marginal gain until the constraint $|\mathbb{A}|=k$ is reached.

## 4. Method

Standard token merging reduces the number of tokens processed in Transformer blocks by identifying and aggregating similar tokens, thereby enabling theoretical speedups proportional to the merge ratio and the model's computational complexity (see Appendix for analysis). It works by selecting destination tokens from the full token set and merging nearby tokens into them based on similarity scores. During the unmerge step, the values of the merged tokens are redistributed to their original positions, preserving fidelity.

Our lightweight and efficient framework, ToMA, improves upon standard token merge at three key stages: 1) Destination Token Selection – efficiently identifying the most representative tokens to serve as merge targets; 2) Token Merge – performing merge operations as a linear transformation, guided by similarity scores computed via attention; 3) Token Unmerge – restoring merged tokens after passing through core computational modules (e.g., Attention, MLP) to original positions through reversed linear transformation.

To achieve further speedups, ToMA a) exploits the spatial locality of the latent space to parallelize operations within local regions and, b) shares merge-related computations across layers and iterations to reduce runtime overhead.

### 4.1. Submodular-Based Destination Selection

Let $\boldsymbol{S}$ be the cosine similarity matrix between all hidden states $\boldsymbol{X}$, where the $\boldsymbol{S}_{ij}$ entry represents the similarity between the $i$ th token and the $j$ th token, namely $\boldsymbol{S}_{ij} \equiv \cos(\boldsymbol{X}_i, \boldsymbol{X}_j)$. We denote the set of all tokens as $\mathbb{V}$ (ground set in submodular optimization) and the set of chosen destination tokens as $\mathbb{D}$.

$$f_{\text{FL}}(\mathbb{D}) = \sum_{\boldsymbol{v}_i\in\mathbb{V}} \max_{\boldsymbol{v}_j\in\mathbb{D}} \boldsymbol{S}_{ij} \tag{2}$$

The submodular function used for destination token selection is the Facility Location function (FL), as shown in Eq. 2. $f_{\text{FL}}(\cdot)$ quantifies how well a subset $\mathbb{D}$ of destination tokens represents the full token set $\mathbb{V}$ by summing, for each token $\boldsymbol{v}_i \in \mathbb{V}$, the maximum similarity $\boldsymbol{S}_{ij}$ to any destination token $\boldsymbol{v}_j \in \mathbb{D}$. Intuitively, this corresponds to asking: for each token in the ground set, how well does the selected subset represent it? A higher value of $f_{\text{FL}}(\mathbb{D})$ implies that every token in $\mathbb{V}$ is closely matched by a representative in $\mathbb{D}$, making $\mathbb{D}$ a compact and diverse summary of the input. This naturally aligns with the objective of token merging, where we aim to preserve global semantic structure using a reduced set of tokens. Notably, our framework is modular—other submodular functions can be substituted for $f_{\text{FL}}$ to customize the selection behavior.

The submodular nature of $f_{\text{FL}}$ provides a theoretical guarantee: greedy maximization yields a near-optimal subset with provably minimal information loss, as discussed in Sec. 3.2. When optimizing the destination set $\mathbb{D}$ using the greedy algorithm (Alg. 1), we iteratively select the token that provides the largest marginal gain $f_{\text{FL}}(\boldsymbol{v}|\mathbb{D}')$ with respect to the current set $\mathbb{D}'$. This marginal gain can be efficiently computed (see Appendix A.1 for derivation) as:

$$\arg\max_{\boldsymbol{v}_i\notin\mathbb{D}'} \sum_{j=1}^{N} \max\left(0, \boldsymbol{S}_{ij} - \boldsymbol{m}_j(\mathbb{D}')\right),$$

where $\boldsymbol{m}_j(\mathbb{D}') = \max_{\boldsymbol{v}_k\in\mathbb{D}'} \boldsymbol{S}_{j,k}$ is a cached vector that stores, for each token $\boldsymbol{v}_j \in \mathbb{V}$, the maximum similarity to any token currently in $\mathbb{D}'$. This caching enables efficient updates: after adding a new token to $\mathbb{D}'$, $\boldsymbol{m}$ can be incrementally updated in constant time per token. Importantly, all these operations—computing similarities, caching, and evaluating marginal gains—can be expressed in matrix form, supported by our derivation in Appendix A.1, making them highly suitable for parallel execution on GPUs. We include our efficient GPU implementation for the greedy algorithm in Appendix A.2. Also, though the iterative nature of submodular optimization is inherently unavoidable, we manage to parallelize this process by breaking it into smaller ground sets in Sec. 4.3.

While there exist more advanced submodular maximization methods such as the Lazier-than-Lazy Greedy algorithm (Mirzasoleiman et al., 2015), their reliance on operations like random subset sampling introduces irregular memory access patterns that are inefficient on GPUs. Therefore, the standard greedy approach strikes a practical balance between solution quality and hardware efficiency.

### 4.2. (Un)merge with Attention

We begin by formulating token merge in its exact form and then show how it naturally generalizes to a linear projection over the input token space, paving the way for our proposed attention-like merging.

Let $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ denote the input matrix of $N$ token em-

beddings, each of dimension $d$. Suppose we select a set of $\mathbb{D}$ destination tokens with indices $\mathcal{D} = \{j_1, \ldots, j_D\} \subseteq \{1, \ldots, N\}$, and partition the input tokens into $D$ disjoint groups:

$$\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_D,$$

$$\text{s.t.} \quad \bigcup_{k=1}^{D} \mathcal{G}_k = \{1, \ldots, N\}, \quad \mathcal{G}_k \cap \mathcal{G}_l = \emptyset \quad \text{for } k \neq l.$$

Each merged token $\boldsymbol{x}_k^{\text{merged}} \in \mathbb{R}^d$ is computed by aggregating the tokens assigned to group $\mathcal{G}_k$. In the simplest case, this is done via uniform averaging:

$$\boldsymbol{x}_k^{\text{merged}} = \sum_{i \in \mathcal{G}_k} \frac{1}{|\mathcal{G}_k|} \boldsymbol{x}_i,$$

or more generally, we allow token-specific weights $\alpha_{k,i}$ with normalization:

$$\boldsymbol{x}_k^{\text{merged}} = \sum_{i \in \mathcal{G}_k} \frac{\alpha_{k,i}}{Z_k} \boldsymbol{x}_i, \quad \text{where} \quad Z_k = \sum_{i \in \mathcal{G}_k} \alpha_{k,i}.$$

This formulation can be unified by expressing the merged tokens as a linear projection over the input matrix:

$$\boldsymbol{X}_{\text{merged}} = \boldsymbol{W}\boldsymbol{X} \in \mathbb{R}^{D \times d},$$

where $\boldsymbol{W} \in \mathbb{R}^{D \times N}$ is a non-negative weight matrix with $W_{ik}$ indicating the contribution of token $k$ to destination $i$. This subsumes both hard merging schemes (e.g., ToMeSD, where each row of $\boldsymbol{W}$ is one-hot) and soft merging (ToMA, where $\boldsymbol{W}$ contains normalized attention scores).

This linear formulation not only provides a principled interpretation of token merging but also enables efficient implementation and reuse of merge/unmerge operations (namely the weight matrix $\boldsymbol{X}$ across steps and layers.

#### 4.2.1. MERGE

Given the formulation above, naturally, one may think of constructing the merge weight matrix $\boldsymbol{W}$ via attention. Specifically, we treat the destination tokens as queries and all input tokens as keys and values, using Scaled Dot-Product Attention (SDPA) to produce similarity scores. These attention scores serve as soft merge assignments.

Let $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ denote the input token matrix, and let $\boldsymbol{D} \in \mathbb{R}^{D \times d}$ be the destination token matrix, formed by selecting a subset of $D$ token embeddings from $\boldsymbol{X}$. We compute an attention matrix $\boldsymbol{A} \in \mathbb{R}^{D \times N}$ between destinations (as queries) and all input tokens (as keys) using SDPA with a temperature parameter $\tau$:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{D}\boldsymbol{X}^\top}{\tau}\right).$$

It is important to note that softmax is applied column-wise rather than row-wise, as a token should not be decomposed into components whose aggregate exceeds 100%. By including an extra identity matrix $\boldsymbol{I}$, we obtain an exact form of SDPA (Eq. 1). Note that the included $\boldsymbol{I}$ is functionally redundant and can be omitted in implementation:

$$\boldsymbol{A} = \text{SDPA}\left(\boldsymbol{X}, \boldsymbol{D}, \boldsymbol{I}, \tau\right).$$

To form a proper merge weight matrix, we normalize the attention matrix row-wise:

$$\tilde{\boldsymbol{A}}_{ij} = \frac{\boldsymbol{A}_{ji}}{\sum_k \boldsymbol{A}_{jk}},$$

and finally, the merged token representation can be obtained via matrix multiplication:

$$\boldsymbol{X}_{\text{merged}} = \tilde{\boldsymbol{A}}\boldsymbol{X} \in \mathbb{R}^{D \times d}.$$

Intuitively, this operation softly assigns each token to a set of destination tokens based on similarity. Highly similar source tokens contribute more to the destination representations, while dissimilar ones contribute less.

By reducing merge to matrix multiplications and the SDPA kernel, ToMA scales up with token count easily and effectively receives a free ride from ongoing GPU architecture and ML system improvements for attention, which are increasingly efficient in both computation and memory usage.

#### 4.2.2. UNMERGE

Feeding merged tokens into core computation modules,

$$\boldsymbol{X}' = \text{ATTENTION/MLP}(\boldsymbol{X}) \in \mathbb{R}^{D \times d},$$

the output we get is still $D$, and we restore the original token resolution by applying an approximate inverse of the merge projection. Let $\boldsymbol{X}'$ denote the output of core computational modules, with merged tokens as input. The goal is to reconstruct the full-resolution token matrix $\boldsymbol{X}'_{\text{unmerged}} \in \mathbb{R}^{N \times d}$.

A principled approach is to apply the Moore–Penrose pseudo-inverse:

$$\boldsymbol{X}'_{\text{unmerged}} = \tilde{\boldsymbol{A}}^{+}\boldsymbol{X}' = \tilde{\boldsymbol{A}}^\top(\tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^\top)^{-1}\boldsymbol{X}'.$$

This provides a least-squares reconstruction that minimizes the error between the original and reconstructed tokens, assuming the merge–transform–unmerge process remains approximately linear. However, computing the pseudo-inverse is computationally expensive, requiring matrix decompositions such as SVD or QR.

Fortunately, under certain conditions, a much simpler and more efficient alternative is available:

$$\boldsymbol{X}'_{\text{unmerged}} = \tilde{\boldsymbol{A}}^\top\boldsymbol{X}',$$

which happens whenever $\tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^\top = \boldsymbol{I}_D$, making the pseudo-inverse $\tilde{\boldsymbol{A}}^+ = \tilde{\boldsymbol{A}}^\top(\tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^\top)^{-1}$ collapse to the simple transpose $\tilde{\boldsymbol{A}}^\top$. This is equivalent to saying that the rows of $\tilde{\boldsymbol{A}}$ are orthonormal. In ToMA, this condition is approached in practice. The facility-location selection promotes destination tokens that cover largely disjoint subsets of the source tokens, making rows of $\tilde{\boldsymbol{A}}$ distinct. Moreover, the low attention temperature $\tau$ adopted sharpens the softmax distribution, concentrating each row's mass on a few source tokens and bringing its $\ell_2$-norm close to one.

Together, these properties imply:

$$\tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^\top = \boldsymbol{I}_D + \varepsilon, \qquad \|\varepsilon\| \ll 1,$$

so that

$$(\tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^\top)^{-1} \approx \boldsymbol{I}_D - \varepsilon, \quad \text{and thus} \quad \tilde{\boldsymbol{A}}^+ \approx \tilde{\boldsymbol{A}}^\top.$$

Empirically, $\tilde{\boldsymbol{A}}^\top$ remains competitive against the exact pseudo-inverse. Given this high fidelity and the substantial computational and memory savings, ToMA adopts the transpose-based unmerge $\tilde{\boldsymbol{A}}^\top \boldsymbol{X}'$ as the default method.

### 4.3. Further Speedups

The overhead of ToMA arises from three main sources: 1) selecting destination tokens $\boldsymbol{D}$ through submodular optimization; 2) computing the attention-based (un)merge weight matrices $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{A}}^+$; and 3) applying the merge $\tilde{\boldsymbol{A}}\boldsymbol{X}$ before, and unmerge $\tilde{\boldsymbol{A}}^+\boldsymbol{X}'$ after each module inside the transformer blocks. To further reduce these overheads, we exploit the locality of the feature space, enabling these computations to be performed within localized regions. Additionally, we reduce the frequency of steps 1) and 2) by reusing destination selections and (un)merge matrices across multiple iterations and transformer layers.

#### 4.3.1. Locality-Aware Token Merging

A key oversight in prior work is the *locality structure* of the latent space. As shown in Fig. 3, where we visualize $k$-means clusters of U-ViT hidden states during the generation of a "tomato", the recolored tokens form a rough preview of the generated image. For example, in the early denoising steps, the clusters appear as coarse, blocky color regions that gradually refine into a recognizable tomato.

Because natural images exhibit strong local coherence–each pixel tends to resemble its immediate neighbors–their latents inherit this property. As a result, merging within a small spatial window aggregates highly similar information, preserving global structure while discarding redundancy only, and is therefore as effective as global merging. In terms of selecting destination tokens, restricting the facility location search to local regions ensures diversity within each tile and avoids competition across tiles for the same destination.

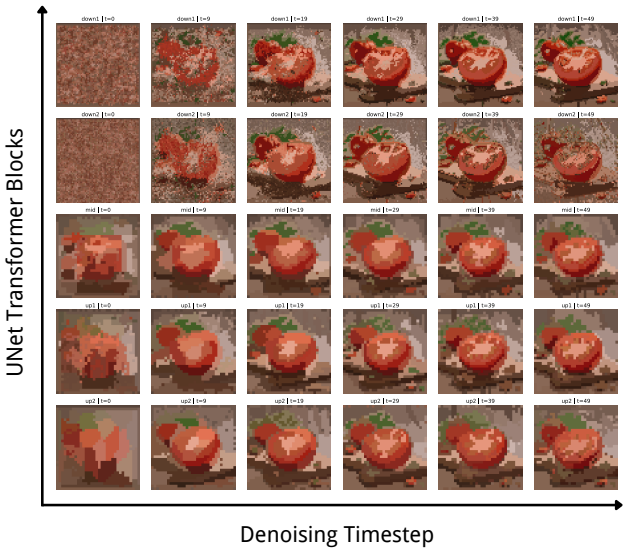


*Figure 3.* Re-colored $k$-means clusters of U-ViT hidden states across transformer blocks and denoising timesteps. A similar visualization on DiT is provided in the Appendix E.1.

The primary benefit of this locality constraint is computational. Splitting the sequence into $k$ equal-sized tiles yields a dominant $1/k$ speed-up for destination selection and an even greater $1/k^2$ reduction in computing the attention weight matrices as well as applying (un)merge. Detailed complexity analysis is provided in Appendix C.

To leverage these benefits, ToMA limits destination selection and (un)merge operations to within localized regions using the two partitioning strategies below.

**Tile-shaped Regions** Tokens are divided into 2-D tiles, preserving both horizontal and vertical proximity. This layout aligns closely with image geometry and gives the best quality, albeit at a reshuffling cost on GPUs.

**Stripe-shaped Regions** Tokens are grouped by rows directly, maintaining memory contiguity and enabling fast reshaping. Although this ignores vertical proximity, it provides the highest speedup.

Both variants substantially reduce computation by operating on smaller subsets in parallel. Tile-shaped regions offer higher fidelity, while stripe-shaped regions run much faster. Despite substantial acceleration, further acceleration is possible by implementing custom tiled/stripe attention kernels, in which memory copying overhead incurred in either reshape or read as strided no longer exists. However, as the post-literature doesn't include such a low-level implementation, we decide to leave it as future work for the fairness of comparison. The full locality-aware ToMA algorithm is given in Appendix Alg. 3 in detail.

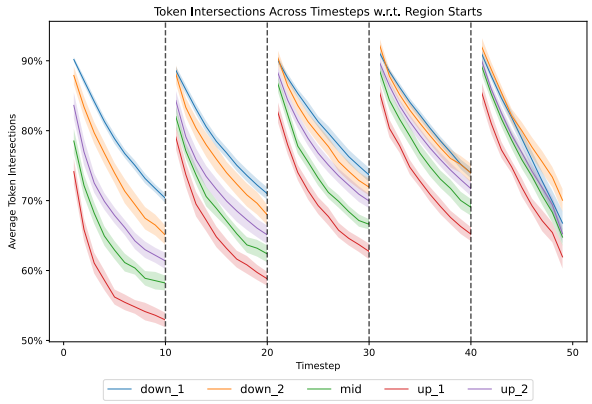


*Figure 4.* Average percentage of shared destination tokens at each denoising timestep relative to the first step of its 10-step interval. Each curve represents a different layer in `SDXL-base` U-ViT model, showing high overlap and gradual divergence over time.

#### 4.3.2. Reusing Destinations and Merge Weights

Hidden states in diffusion models change gradually, so the destination tokens chosen at one step are often similar to those chosen nearby in time. Fig. 4 quantifies this effect: across a 10-step window, more than half of the destinations are reused. Exploiting this redundancy, ToMA reuses the same destination set for several consecutive steps and, because merge and unmerge are linear, also reuses the associated weight matrices across layers. Sharing both the destination selection and the (un)merge matrices sharply reduces the frequency of introducing expensive similarity computation overhead while preserving image quality.

## 5. Experiments

### 5.1. Descriptions

**Setup** We evaluate ToMA on two of the most widely used diffusion models: the UNet-based `SDXL-base` and the DiT-based `Flux.1-dev`, to generate $1024 \times 1024$ images, using the `Diffusers` framework. Importantly, ToMA is architecture-agnostic and can be readily extended to other diffusion models (e.g., `SD2, SD3.5`). Prompts are drawn from the GEMRec dataset (Guo et al., 2024) and ImageNet–1K names of the classes (Deng et al., 2009).

**Metrics** To assess image quality, we use CLIP-T, DINO, and FID (Radford et al., 2021; Caron et al., 2021; Heusel et al., 2017). CLIP-T measures semantic alignment between images and prompts via cosine similarity between image-text embeddings. DINO measures perceptual consistency by comparing visual features between the original generated image and its counterpart produced with the merge method applied. FID (Fréchet Inception Distance) quantifies distributional similarity to real images based on Inception-V3 feature statistics. For FID CLIP-T and DINO in the main experiments shown in this section, we generate 3,000 images and compute scores against ground-truths from ImageNet–1K. For the ablation experiments listed in Appendix F, we generate images from 50 prompts with three random seeds each and report the average. FID is omitted on GEMRec due to the lack of paired images. Inference latency is reported as the median wall-clock time over 100 runs.

**Baselines** We compare ToMA with three heuristic token-reduction baselines: Token Merging for Stable Diffusion (ToMe), Token Downsampling (ToDo), and Token-Fusion (ToFu). These approaches were originally developed for UNet-based architectures and perform well in that setting. As a result, they are not compatible with DiT models, since they lack mechanisms to handle positional embeddings in DiTs. Due to this limitation, we evaluate all three baselines only on `SDXL-base`, leaving `Flux.1-dev` benchmarked solely with ToMA. In terms of implementation, we utilize the official codebases for ToDo and ToMe, and reimplement ToFu based on its paper, as the public code is not available. In our experiment, ToDo uses a fixed merge ratio of 75%, corresponding to a 4-to-1 token downsampling scheme, which represents the lowest merge ratio supported by its implementation.

We additionally report TLB (Theoretical Lower Bound), which approximates the maximum attainable speedup by reducing the number of tokens without incurring extra runtime overhead (e.g., gather tokens). To simulate this bound, we do a dummy merge–directly drop tokens and proceed with the next module by duplicating retained token features to preserve input shape, thereby isolating the theoretical benefits of token reduction while minimizing implementation-specific costs. Quality metrics are omitted for TLB, as cloned tokens do not yield valid outputs for evaluation.

**ToMA Variants** To analyze the impact of locality on destination selection and (un)merge operations, we evaluate four configurations: 1) `ToMA`, our default setting, which uses tile-based destination selection and global attention-based merge; 2) $\texttt{ToMA}_{\texttt{stripe}}$, which restricts both destination and merge operations to within stripe regions; 3) $\texttt{ToMA}_{\texttt{tile}}$, which uses tile regions for both destination and merge; and 4) $\texttt{ToMA}_{\texttt{once}}$, which improves efficiency by performing (un)merge operations only once per Transformer block—at the beginning and end—rather than around each core computation module. As for the hyperparameter, stripe- and tile-based configurations use 64 stripes or tiles, respectively. We reuse the destination for 10 denoising steps and reuse merge weights for 5 steps, with each block of a given type sharing one set. No reuse across denoising timesteps in `Flux.1-dev` but within blocks of the same kind.

## 5.2. Results

| Ratio | Method | Metrics | | | Sec/img ↓ | | |
|---|---|---|---|---|---|---|---|
| | | FID↓ | CLIP-T↑ | DINO↓ | RTX6000 | V100 | RTX8000 |
| **Baseline** | SDXL-base | 25.27 | 29.89 | 0 | 6.1 | 14.5 | 16.1 |
| 0.25 | ToMA | 25.72 | 29.86 | 0.048 | 6.0 | 14.3 | 15.9 |
| | ToMA$_{stripe}$ | 25.17 | 29.90 | 0.054 | 5.6 | 12.6 | 14.5 |
| | ToMA$_{tile}$ | 25.43 | 29.86 | 0.045 | 6.2 | 13.6 | 15.7 |
| | ToMA$_{once}$ | 26.31 | 29.70 | 0.052 | 5.5 | 12.3 | 13.5 |
| | TLB | – | – | – | 5.2 | 12.1 | 9.2 |
| 0.50 | ToMA | 28.88 | 29.64 | 0.068 | 5.0 | 11.0 | 12.8 |
| | ToMA$_{stripe}$ | 29.11 | 29.52 | 0.074 | 4.6 | 10.1 | 12.0 |
| | ToMA$_{tile}$ | 29.19 | 29.63 | 0.063 | 6.3 | 11.1 | 13.2 |
| | ToMA$_{once}$ | 38.14 | 29.06 | 0.080 | 4.9 | 9.7 | 11.5 |
| | TLB | – | – | – | 4.0 | 9.9 | 7.8 |
| 0.75 | ToMA | 58.59 | 27.96 | 0.098 | 4.3 | 8.5 | 9.8 |
| | ToMA$_{stripe}$ | 89.93 | 26.97 | 0.110 | 4.5 | 8.0 | 9.5 |
| | ToMA$_{tile}$ | 58.90 | 28.17 | 0.091 | 6.2 | 9.1 | 10.7 |
| | ToMA$_{once}$ | 123.37 | 24.96 | 0.106 | 4.9 | 7.6 | 8.9 |
| | TLB | – | – | – | 3.1 | 7.8 | 6.5 |

*Table 1.* Performance comparison between ToMA variants and SDXL-base (Baseline) for generating 1024×1024 images over 50 sampling steps. Best values are highlighted, except for TLB. (↑: higher better, ↓: lower better).

**UNet Results** Table 1 shows that ToMA$_{tile}$ consistently performs the best on DINO score, indicating strong perceptual alignment, but is slowed down by low-level memory copying overhead during token tiling. Our manual inspection of generated images also confirms that tile-based merging yields the highest visual quality. In contrast, ToMA$_{stripe}$ benefits from faster runtimes due to its sequential memory access pattern, which enables direct reshaping without copying; however, the absence of a strong locality leads to slightly degraded image quality. ToMA finds a favorable trade-off, achieving up to **24%** speedup and consistent performance across different GPU architectures. Based on this balance between efficiency and quality, we adopt it as our default method. Finally, the experimental variant ToMA$_{once}$ offers the highest acceleration by treating the entire Transformer block as a single merge unit, significantly reducing overhead. However, it produces the worst quality due to insufficient spatial-context mixing across layers. Interestingly, in some scenarios, its runtime is even lower than that of the TLB, likely due to less frequent memory copying compared to our dummy merge implementation.

**DiT Results** For the Flux model, we include only ToMA and ToMA$_{tile}$, as stripe-based merging is incompatible with the rotary positional embedding (RoPE) used in Flux. We also exclude results on the V100 GPU due to out-of-memory (OOM) failures. As shown in Tab. 2, both ToMA variants consistently accelerate generation across GPUs, with ToMA achieving up to a 23.4% speedup without compromising image quality when merging down to 50% of tokens. Although ToMA$_{tile}$ is marginally slower due to memory overhead, it consistently offers better fidelity, evidenced by stronger CLIP-T and DINO scores. This efficiency–quality tradeoff supports our choice of ToMA as the default, while also highlighting the need for optimized low–level implementations to fully realize the benefits of locality-aware merging.

| Ratio | Method | Metrics | | | RTX8000 | | RTX6000 | |
|---|---|---|---|---|---|---|---|---|
| | | FID↓ | CLIP-T↑ | DINO↓ | Sec/img | ↓Δ | Sec/img | ↓Δ |
| **Baseline** | Flux.1-dev | 31.56 | 29.03 | 0 | 59.20 | 0% | 21.03 | 0% |
| 0.25 | ToMA | 30.80 | 29.07 | 0.043 | 56.70 | –4.2% | 20.14 | –4.2% |
| | ToMA$_{tile}$ | 31.49 | 29.05 | 0.021 | 57.47 | –2.9% | 20.78 | –1.2% |
| 0.50 | ToMA | 31.70 | 29.09 | 0.051 | 51.44 | –13.1% | 18.58 | –11.6% |
| | ToMA$_{tile}$ | 32.95 | 29.19 | 0.032 | 53.61 | –9.4% | 19.61 | –6.8% |
| 0.75 | ToMA | 33.39 | 28.98 | 0.064 | 49.83 | –15.9% | 16.12 | –23.4% |
| | ToMA$_{tile}$ | 33.88 | 29.34 | 0.045 | 49.86 | –15.8% | 18.30 | –12.9% |

*Table 2.* Performance comparison of ToMA variants and Flux.1-dev (Baseline) for 1024×1024 image generation (35 sampling steps). Best values are highlighted, and relative speed improvements (Δ) are shown as %. Negative Δ values indicate faster inference compared to the baseline (lower is better).

**Benchmark** Table 3 presents a comparative evaluation of ToMA (our default method), alongside other token reduction strategies on the SDXL model. Among all methods, ToMA achieves the fastest generation time—up to 28.5% improvement—while maintaining competitive perceptual quality, making it the most balanced choice. Although ToMe (Token Merge) delivers slightly better image quality, as reflected in CLIP-T and DINO scores, it suffers from severe latency due to its complex token selection mechanism, which relies on GPU-inefficient operations (e.g., sort). ToFu (Token Fusion), while faster than ToMe with lower overhead thanks to its blended strategy that incorporates token pruning, exhibits unstable visual quality—some generations appear acceptable, but others are heavily degraded, especially at higher reduction ratios. The high FID values confirm this observation. Lastly, ToDo (Token Downsampling) offers moderate gains in both speed and fidelity but demonstrates noticeable distributional shifts, as indicated by its elevated FID

| Ratio | Method | FID↓ | CLIP-T↑ | DINO↓ | Sec/img↓ | ↓Δ |
|---|---|---|---|---|---|---|
| **Baseline** | SDXL | 25.3 | 29.89 | 0 | 6.07 | 0% |
| 0.25 | ToMA | 25.7 | 29.86 | 0.048 | 6.03 | –0.7% |
| | ToMe | 25.6 | 29.86 | 0.054 | 8.66 | +42.7% |
| | ToFu | 35.2 | 29.34 | 0.072 | 6.92 | +14.0% |
| 0.50 | ToMA | 28.9 | 29.64 | 0.068 | 5.04 | –17.0% |
| | ToMe | 26.7 | 29.71 | 0.071 | 8.73 | +43.8% |
| | ToFu | 142.1 | 25.04 | 0.134 | 6.83 | +12.5% |
| 0.75 | ToMA | 58.6 | 27.96 | 0.098 | 4.34 | –28.5% |
| | ToMe | 41.2 | 29.09 | 0.084 | 8.16 | +34.4% |
| | ToFu | 161.5 | 24.13 | 0.148 | 6.76 | +11.4% |
| | ToDo | 68.6 | 27.60 | 0.093 | 5.67 | –6.6% |

*Table 3.* Performance comparison of ToMA, SDXL-base (Baseline), and other token reduction methods for generating 1024×1024 images (50 sampling steps). Best values are highlighted, and relative speed improvements (Δ) are shown as %. (−Δ faster, +Δ slower; ↑: higher better, ↓: lower better).

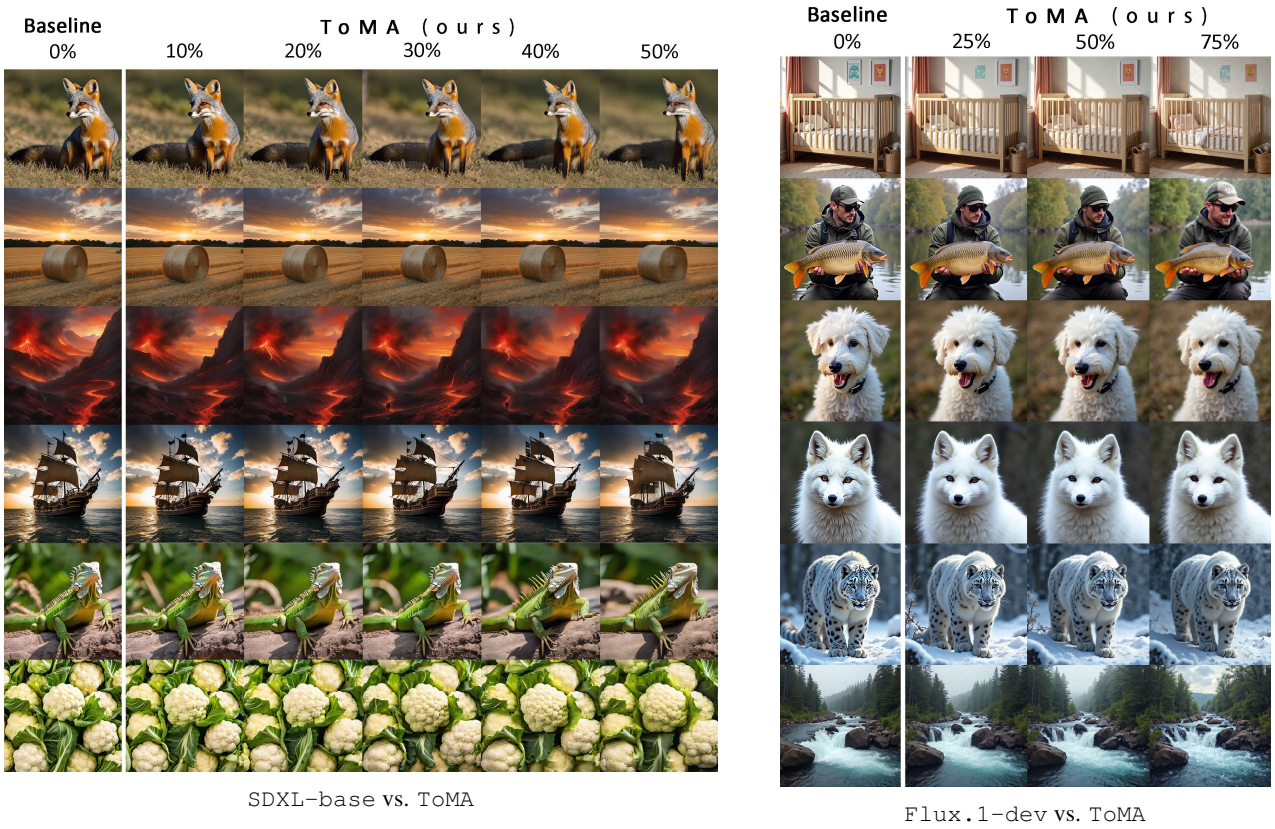


*Figure 5.* Visual comparison of baselines (SDXL & Flux) versus (ToMA) under different higher token-merge ratios. Despite merging up to 50% of tokens, ToMA preserves sharp details and overall scene coherence. Additional qualitative examples are provided in Appendix D.

score. Hwang et al. discuss the possible cause that the noise reduction due to downsampling creates a difference in the signal-to-noise ratio that leads to suboptimal performance if applied directly (2024). Overall, these results highlight ToMA as the most robust and well-rounded token merging strategy for efficient high-resolution image generation.

**Qualitative Result** Figure 5 provides a side-by-side visual comparison of images generated by the original models versus our ToMA-accelerated variants at several token-merge ratios. On the left, six different prompts are shown for SDXL and ToMA at 10% – 50% token reduction. Even as the merge ratio increases, ToMA's outputs (second through sixth columns) remain nearly indistinguishable from the originals. Similarly, on the right, we present six different prompts for Flux vs. ToMA at 25%, 50%, and 75% token reduction. Even at 75% merging, ToMA's results (fourth column) faithfully reproduce key details.

**Ablation & Otheres.** ToMA's robustness is further validated by comprehensive ablations (Appendix F), which investigate the impact of merge frequency, tile/stripe granularity, unmerge strategies (e.g., transpose vs. pseudo-inverse), and sharing schedules. These studies confirm our design choices, such as selecting tile-based merge with 256 tiles and transpose-based unmerge for optimal speed–quality balance. Appendix G provides memory profiling across multiple models and sparsity levels, showing that ToMA variants incur negligible memory overhead compared to dense baselines. Appendix H complements this by providing a detailed FLOP analysis, saving computations up to $3.4\times$.

## 6. Conclusions

In this work, we introduce Token Merge with Attention for Diffusion Models (ToMA), a co-designed merging framework that advances prior methods in three key aspects: 1) more representative token selection via submodular optimization with theoretical guarantee; 2) a flexible and efficient merge–unmerge mechanism implemented through attention-based operations; and 3) the incorporation of locality-aware and shared merging strategies to maximize runtime gains. Together, these improvements yield substantial speedups in practice while preserving high image fidelity across different GPU and model architectures (U-ViT & DiT), establishing ToMA as a robust and deployable solution for efficient high-resolution generation.

## Acknowledgements

We gratefully acknowledge the New York University High Performance Computing (NYU HPC) facility for providing the GPU clusters and technical support that enabled the experiments in this work.

## Impact Statement

Our work on Token Merge with Attention (ToMA) improves the efficiency of diffusion models for image generation. While this advancement democratizes access to high-quality AI art creation, it's important to acknowledge that such technologies can be misused to generate misleading content or deepfakes. Additionally, as these models are trained on large internet-scraped datasets, there's a risk of perpetuating societal biases in the training data. We recognize these ethical considerations and emphasize the importance of responsible development and use of such technologies.

## References

Bao, F., Nie, S., Xue, K., Cao, Y., Li, C., Su, H., and Zhu, J. All are worth words: A vit backbone for diffusion models. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 22669–22679, 2023.

Bolya, D. and Hoffman, J. Token merging for fast stable diffusion, 2023. URL https://arxiv.org/abs/2303.17604.

Bolya, D., Fu, C.-Y., Dai, X., Zhang, P., Feichtenhofer, C., and Hoffman, J. Token merging: Your ViT but faster. In *ICLR*, 2023.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. Emerging properties in self-supervised vision transformers. *CoRR*, abs/2104.14294, 2021. URL https://arxiv.org/abs/2104.14294.

Chen, M., Shao, W., Xu, P., Lin, M., Zhang, K., Chao, F., Ji, R., Qiao, Y., and Luo, P. Diffrate: Differentiable compression rate for efficient vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17164–17174, 2023.

Dao, T. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*, 2023.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Dhariwal, P. and Nichol, A. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems (NIPS/NeurIPS)*, 34:8780–8794, 2021.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. URL https://arxiv.org/abs/2010.11929.

Fayyaz, M., Koohpayegani, S. A., Jafari, F. R., Sengupta, S., Joze, H. R. V., Sommerlade, E., Pirsiavash, H., and Gall, J. Adaptive token sampling for efficient vision transformers. In *ECCV*, pp. 396–414, 2022.

Fujishige, S. *Submodular functions and optimization*. Elsevier, 2005.

Guo, Y., Liu, H., and Wen, H. Gemrec: Towards generative model recommendation. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, volume 9 of *WSDM '24*, pp. 1054–1057. ACM, March 2024. doi: 10.1145/3616855.3635700. URL http://dx.doi.org/10.1145/3616855.3635700.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *NeurIPS*, 30, 2017.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems (NIPS/NeurIPS)*, 33:6840–6851, 2020.

Hwang, J., Park, Y.-H., and Jo, J. Upsample guidance: Scale up diffusion models without training. *arXiv preprint arXiv:2404.01709*, 2024.

Kim, M., Gao, S., Hsu, Y.-C., Shen, Y., and Jin, H. Token fusion: Bridging the gap between token pruning and token merging, 2023. URL https://arxiv.org/abs/2312.01026.

Kim, S., Shen, S., Thorsley, D., Gholami, A., Kwon, W., Hassoun, J., and Keutzer, K. Learned token pruning for transformers, 2022. URL https://arxiv.org/abs/2107.00910.

Lefaudeux, B., Massa, F., Liskovich, D., Xiong, W., Caggiano, V., Naren, S., Xu, M., Hu, J., Tintore, M., Zhang, S., Labatut, P., Haziza, D., Wehrstedt, L., Reizenstein, J., and Sizov, G. xformers: A modular and hackable transformer modelling library. https://github.com/facebookresearch/xformers, 2022.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. Swin transformer: Hierarchical vision transformer using shifted windows. In *ICCV*, pp. 10012–10022, 2021a.

Liu, Z., Wang, Y., Han, K., Ma, S., and Gao, W. Post-training quantization for vision transformer. *arXiv preprint arXiv:2106.14156*, 2021b.

Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A., Vondrák, J., and Krause, A. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Nemhauser, G. L., Wolsey, L. A., and Fisher, M. L. An analysis of approximations for maximizing submodular set functions—i. *Mathematical programming*, 14:265–294, 1978.

Papa, L., Russo, P., Amerini, I., and Zhou, L. A survey on efficient vision transformers: Algorithms, techniques, and performance benchmarking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):7682–7700, 2024. doi: 10.1109/TPAMI.2024.3392941.

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

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

Rao, Y., Zhao, W., Liu, B., Lu, J., Zhou, J., and Hsieh, C.-J. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *NeurIPS*, November 2021.

Smith, E., Saxena, N., and Saha, A. Todo: Token downsampling for efficient generation of high-resolution images. *arXiv preprint arXiv:2402.13573*, 2024.

Song, Y., Sohl-Dickstein, J., Kingma, D. P., Kumar, A., Ermon, S., and Poole, B. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021. URL https://openreview.net/forum?id=PxTIG12RRHS.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. Training data-efficient image transformers & distillation through attention. In *ICML*, pp. 10347–10357, 2021.

von Platen, P., Patil, S., Lozhkov, A., Cuenca, P., Lambert, N., Rasul, K., Davaadorj, M., Nair, D., Paul, S., Berman, W., Xu, Y., Liu, S., and Wolf, T. Diffusers: State-of-the-art diffusion models. https://github.com/huggingface/diffusers, 2022.

Wang, W., Xie, E., Li, X., Fan, D.-P., Song, K., Liang, D., Lu, T., Luo, P., and Shao, L. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *ICCV*, pp. 568–578, 2021.

Yin, H., Vahdat, A., Alvarez, J. M., Mallya, A., Kautz, J., and Molchanov, P. A-ViT: Adaptive tokens for efficient vision transformer. In *CVPR*, pp. 10809–10818, June 2022a.

Yin, H., Vahdat, A., Alvarez, J. M., Mallya, A., Kautz, J., and Molchanov, P. A-vit: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10809–10818, 2022b.

Yu, L. and Xiang, W. X-pruner: Explainable pruning for vision transformers. In *CVPR*, pp. 24355–24363, 2023.

# A. Facility Location for Selecting Destination Tokens

## A.1. Mathematical Foundations for Efficient Greedy Maximization

We begin by defining the notations used in the greedy selection procedure. Let $\mathbb{V}$ denote the full set of tokens, and let $\mathbb{A} \subseteq \mathbb{V}$ be the current set of selected representative tokens. At each step, the greedy algorithm selects the next token $\boldsymbol{v}^* \in \mathbb{V} \setminus \mathbb{A}$ that maximizes the marginal gain:

$$\boldsymbol{v}^* = \arg\max_{\boldsymbol{v} \in (\mathbb{V} \setminus \mathbb{A})} f(\boldsymbol{v}|\mathbb{A}),$$

where the gain function $f(\boldsymbol{v}|\mathbb{A})$ is defined as the increase in a coverage objective when $\boldsymbol{v}$ is added to $\mathbb{A}$. Formally,

$$\begin{aligned} f(\boldsymbol{v}|\mathbb{A}) &= f(\mathbb{A}') - f(\mathbb{A}) \\ &= \sum_{\boldsymbol{u} \in \mathbb{V}} \max_{\boldsymbol{u}_a \in \mathbb{A}'} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_a) - \sum_{\boldsymbol{u} \in \mathbb{V}} \max_{\boldsymbol{u}_b \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_b), \quad \text{where } \mathbb{A}' = \{\boldsymbol{v}\} \cup \mathbb{A}. \end{aligned}$$

Here, $\text{sim}(\cdot, \cdot)$ denotes the similarity between two tokens (we use cosine similarity in practice for computational efficiency). The first term in the equation measures for each token $\boldsymbol{v} \in \mathbb{V}$, how well it is represented in the updated set $\mathbb{A}' = \{\boldsymbol{v}\} \cup \mathbb{A}$. The second term asks the same question but with the current set $\mathbb{A}$. Their difference quantifies the marginal gain of including $\boldsymbol{v}$ in the representative set.

We can simplify the first term in the gain function by observing that:

$$\sum_{\boldsymbol{u} \in \mathbb{V}} \max_{\boldsymbol{u}_a \in \mathbb{A}'} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_a) = \sum_{\boldsymbol{u} \in \mathbb{V}} \max \left\{ \max_{\boldsymbol{u}_a \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_a), \text{sim}(\boldsymbol{u}, \boldsymbol{v}) \right\},$$

in which the identity holds because for each $\boldsymbol{u} \in \mathbb{V}$, the maximum similarity with the updated set $\mathbb{A}'$ is either its similarity with $\boldsymbol{v}$, or its previous maximum over $\mathbb{A}$. So, we take the maximum of the two.

Substituting this into the definition of $f(\boldsymbol{v}|\mathbb{A})$, we obtain:

$$\begin{aligned} f(\boldsymbol{v}|\mathbb{A}) &= \sum_{\boldsymbol{u} \in \mathbb{V}} \left( \max \left\{ \max_{\boldsymbol{u}_a \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_a), \text{sim}(\boldsymbol{u}, \boldsymbol{v}) \right\} - \max_{\boldsymbol{u}_b \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_b) \right) \\ &= \sum_{\boldsymbol{u} \in \mathbb{V}} \max \left\{ 0, \text{sim}(\boldsymbol{u}, \boldsymbol{v}) - \max_{\boldsymbol{u}_b \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_b) \right\}. \end{aligned}$$

This simplification shows that the marginal gain of adding $\boldsymbol{v}$ depends only on the tokens $\boldsymbol{u} \in \mathbb{V}$ for which $\boldsymbol{v}$ provides a new maximum similarity beyond what is already achieved by the current set $\mathbb{A}$.

As a result, the greedy selection objective becomes:

$$\boldsymbol{v}^* = \arg\max_{\boldsymbol{v} \in (\mathbb{V} \setminus \mathbb{A})} \sum_{\boldsymbol{u} \in \mathbb{V}} \max \left\{ 0, \text{sim}(\boldsymbol{u}, \boldsymbol{v}) - \max_{\boldsymbol{u}_b \in \mathbb{A}} \text{sim}(\boldsymbol{u}, \boldsymbol{u}_b) \right\}.$$

This formulation enables efficient computation of the marginal gain for each candidate token and facilitates the selection of the token that most improves the representational coverage of the current set.

### A.2. Facility Location Algorithm

This algorithm implements a greedy approach to select $D$ tokens based on the Facility Location objective. It begins by computing the sum of similarities for each token. The first token chosen is the one with the highest sum of similarities. Then, it iteratively selects the remaining $D-1$ tokens.

In each iteration, the algorithm computes its gain for every unselected token, representing the overall similarity improvement. The token with the highest gain is chosen and added to the selection. After each selection, the algorithm updates the maximum similarities achieved so far. This process continues until $D$ tokens are selected.

By selecting tokens that maximize the marginal gain in similarity at each step, this approach effectively covers the input space while avoiding redundancy, ensuring a diverse and representative set of tokens.

**Algorithm 2:** Greedy Algorithm for Token Selection

**Input:** Similarity matrix $\boldsymbol{S} \in \mathbb{R}^{N \times N}$, number of tokens to select $D$
**Output:** Selected destination token indices $\boldsymbol{d}$
**Initialize**: $\boldsymbol{d} \leftarrow \{\}$;
Sum over each row: $\boldsymbol{s} = \sum_{j=1}^{N} \boldsymbol{S}_{ij}$;
Select the first token index: $\boldsymbol{t}_1 \leftarrow \arg\max_i \boldsymbol{s}_i$;
Add the greedy choice to destination tokens: $\boldsymbol{d} \leftarrow \boldsymbol{d} \cup \{\boldsymbol{t}_1\}$;
Create the cache vector $\boldsymbol{m}_j(\mathbb{D}')\big)$ from the corresponding row in the $\boldsymbol{S}$: $\boldsymbol{m}_j(\mathbb{D}')\big) \leftarrow \boldsymbol{S}_{\boldsymbol{t}_1}$;
Set to zero to avoid re-selection: $\boldsymbol{S}_{\boldsymbol{t}_1} \leftarrow 0$;
**for** $k = 2$ **to** $D$ **do**
  **for** *each token index* $i$ *not in* $\boldsymbol{d}$ **do**
    Compute the marginal gain efficiently with cache: $\boldsymbol{g} = \sum_{j=1}^{N} \max\{0, \boldsymbol{S}_{ij} - \boldsymbol{m}_j(\mathbb{D}')\}$;
  **end**
  Select the next token greedily: $\boldsymbol{t}_k \leftarrow \arg\max_{i \notin \boldsymbol{d}} \boldsymbol{g}_i$;
  Add the newly selected token index: $\boldsymbol{d} \leftarrow \boldsymbol{d} \cup \{\boldsymbol{t}_k\}$;
  Update largest row: $\boldsymbol{m}_j(\mathbb{D}') \leftarrow \max\{\boldsymbol{m}_j(\mathbb{D}'), \boldsymbol{S}_{\boldsymbol{t}_k}\}$;
  Set to zero to avoid re-selectoin: $\boldsymbol{S}_{\boldsymbol{t}_k} \leftarrow 0$;
**end**
**return** $\boldsymbol{d}$

## B. Overall Detailed Algorithm of ToMA

**Algorithm 3:** ToMA with Local Regions

**Input:** Tensor $\mathbf{X} \in \mathbb{R}^{B\times N\times d}$ (input sequence), $D$ (number of destination tokens), $\tau$ (attention temperature), $F(\cdot)$ (core computational module (e.g., MLP, Attention)

**Split into local regions**

Partition the sequence dimension into $P$ blocks, $\mathbf{X} \leftarrow (\mathbf{X}_1, \ldots, \mathbf{X}_P)$ with $\mathbf{X}_p \in \mathbb{R}^{B\times N_{\text{loc}}\times d}$ and $N_{\text{loc}}P = N$;

$D_{\text{loc}} \leftarrow D/P$;

$\mathbf{X} \leftarrow \mathbf{X}.\text{reshape}(B\cdot P,\, N_{\text{loc}},\, d)$;

**Step 1: Facility–location token selection**

$(T_1, \ldots, T_{B\cdot P}) \leftarrow \text{Greedy}(f_{\text{FL}},\, D_{\text{loc}},\, \mathbf{X})$;

$\mathbf{X}_T \leftarrow (\mathbf{X}_{1,T_1}, \ldots, \mathbf{X}_{B\cdot P,T_{B\cdot P}}) \in \mathbb{R}^{BP\times D_{\text{loc}}\times d}$;

**Step 2: Merge**

$\mathbf{A} \leftarrow \text{SDPA}(\mathbf{X}_T,\, \mathbf{X},\, \mathbf{I},\, \tau) \in \mathbb{R}^{BP\times D_{\text{loc}}\times N_{\text{loc}}}$;

$\tilde{\mathbf{A}} \leftarrow \mathbf{A} \,/\, \mathbf{A}\textstyle\sum_{-1}$;

$\mathbf{X}_{\text{merged}} \leftarrow \tilde{\mathbf{A}}\,\mathbf{X} \in \mathbb{R}^{BP\times D_{\text{loc}}\times d}$;

**Computational layer**

$\mathbf{X}' \leftarrow F\big(\mathbf{X}_{\text{merged}}.\text{reshape}(B,\, D,\, d)\big)$;

**Step 3: Unmerge**

$\mathbf{X}'_{\text{unmerged}} \leftarrow \tilde{\mathbf{A}}^{\top}\,\mathbf{X}'$;

Reassemble $\mathbf{X}'_{\text{unmerged}}$ to reverse the local-region split;

**return** $\mathbf{X}'_{\text{unmerged}}$

**Description.** Algorithm 3 details a single ToMA layer equipped with local-region processing. Given an input tensor $\mathbf{X}\in\mathbb{R}^{B\times N\times d}$, we first shard the sequence into $P$ equally sized local regions of length $N_{\text{loc}}$ (lines 1–3), then assign a budget of $D_{\text{loc}}$ *destination tokens* to each region (line 2).

In **Step 1** we invoke a GPU-friendly greedy facility–location algorithm to pick, for every mini-batch $\times$ region pair, the set of tokens whose neighbourhoods best cover the local region (lines 5–6). These destinations are gathered into $\mathbf{X}_T$.

**Step 2** forms a scaled-dot-product attention map $\mathbf{A}$ from each destination to *all* tokens in its region, row-normalises it to $\tilde{\mathbf{A}}$, and left-multiplies $\mathbf{X}$ to obtain the merged representation $\mathbf{X}_{\text{merged}}$ whose sequence length is reduced from $N$ to $D$ (lines 8–10). Any computational block $F(\cdot)$—e.g., a transformer layer or UNet block—can now process the shorter sequence (line 12), yielding a computed reduction factor of $N/D$ without altering the block's internal parameters.

Now, in **Step 3**, we distribute the updated destination embeddings back to their original token positions via $\tilde{\mathbf{A}}^{\top}$ and undoes the initial region partitioning (lines 14–15), so that the next layer receives a full-resolution tensor. Because the (un)merge operations are purely linear and share weights across the batch, their overhead is negligible, and they can be fused with existing attention kernels. Repeating this procedure layer-by-layer yields significant wall-clock speed-ups while exhibiting scarcely any perceptible degradation in image quality.

## C. Computational–complexity analysis

We retain explicit constant factors in the flop count because they translate directly to empirical speedups on modern GPUs. Throughout, $N$ is the original sequence length, $D$ the length *after* token merging, $d$ the embedding dimension, and

$$r \;=\; \frac{D}{N} \qquad \text{(merge ratio, i.e. the fraction of tokens } \textit{kept}\text{)}.$$

**Baseline self–attention block.** Treating each matrix multiplication as a collection of dot products, the total number of scalar multiplications for a standard self–attention block is

$$C_{\text{base}} \;=\; 4\,d^2 N \;+\; 2\,d\,N^2.$$

The first term ($Q, K, V$ projections and the output projection) scales linearly in $N$; the second term ($QK^\top$ and attention-value product) scales quadratically.

**Token–merged self–attention.** After reducing the token count from $N$ to $D = rN$, the attention cost becomes

$$C_{\text{attn}}(D) \;=\; 4\,d^2 D \;+\; 2\,d\,D^2 \;=\; 4\,d^2 rN \;+\; 2\,d\,r^2 N^2.$$

Hence the *ideal* speedup (ignoring merge overhead) is

$$\text{Speedup}_{\text{ideal}} \;=\; \frac{C_{\text{base}}}{C_{\text{attn}}(D)} \;=\; \frac{4d + 2N}{4d\,r + 2N\,r^2}.$$

**Overheads introduced by ToMA.** Token merging incurs several additional costs:

- **Submodular–selection overhead** (computing marginal gains for all pairs): $C_{\text{sub}} = N^2 d$.
- **Merge–attention projection** (computing pairwise weights): $C_{\text{proj}} = NDd$.
- **Merge operation** (applying the computed weights to produce merged tokens): $C_{\text{merge}} = NDd$.
- **Unmerge operation** with a transpose-style redistribution: $C_{\text{unmerge}} = NDd$.

Summing the three linear–in-$D$ overhead terms yields $C_{\text{lin}} = 3NDd = 3rN^2 d$.

**Total cost with ToMA.** The overall computational cost after merging is therefore

$$C_{\text{total}}(r) \;=\; \underbrace{4\,d^2 rN + 2\,d\,r^2 N^2}_{\text{attention block}} \;+\; \underbrace{N^2 d}_{\text{submodular}} \;+\; \underbrace{3rN^2 d}_{\text{merge / unmerge}}.$$

**Realistic speedup.** The practical speedup of ToMA relative to the baseline is

$$\text{Speedup}_{\text{practical}} \;=\; \frac{C_{\text{base}}}{C_{\text{total}}(r)} \;=\; \frac{4dN + 2N^2}{4drN + N^2\big(1 + 3r + 2r^2\big)}.$$

**Discussion.** For practical transformer settings we have $N \gg d$, so the quadratic terms $2d\,r^2 N^2$ (remaining attention cost) and $N^2 d$ (one–shot sub-modular selection) dominate. Consequently, the empirical speedup is well approximated by $\text{Speedup} \;\approx\; \big(2 + 4d/N\big)/\big(2r^2 + 1 + 3r\big)$, which approaches the analytic bound $\frac{2}{2r^2+1}$ whenever $r \lesssim 0.5$ and $d/N$ is small. At moderate merge ratios ($r \in [0.25, 0.5]$) this yields the $24-28\%$ latency drop we observe on SDXL/Flux. If we push $r$ below $\sim 0.1$, the linear–in–$r$ merge overhead $3rN^2 d$ and the fixed $N^2 d$ selection cost start to dominate, so further merging brings diminishing returns—underscoring why our locality-aware pattern reuse amortises these costs across multiple layers.

# D. More Qualitative Results

## D.1. Images Generated with SDXL

We present additional visual comparisons between `SDXL`, ToMeSD, and ToMA below in Fig 6. The prompts are sampled from the GemRec and ImageNet-1K datasets. As the visuals reveal, ToMA maintains image quality more faithfully compared to other methods, especially in preserving fine-grained details and spatial coherence. This is evident in both synthetic scenes (e.g., fantasy landscapes, bowls of fire) and natural subjects (e.g., animals, boats, and portraits). ToMA Additional images generated with ToMA on `SDXL` are provided on the next page in Fig. 7.

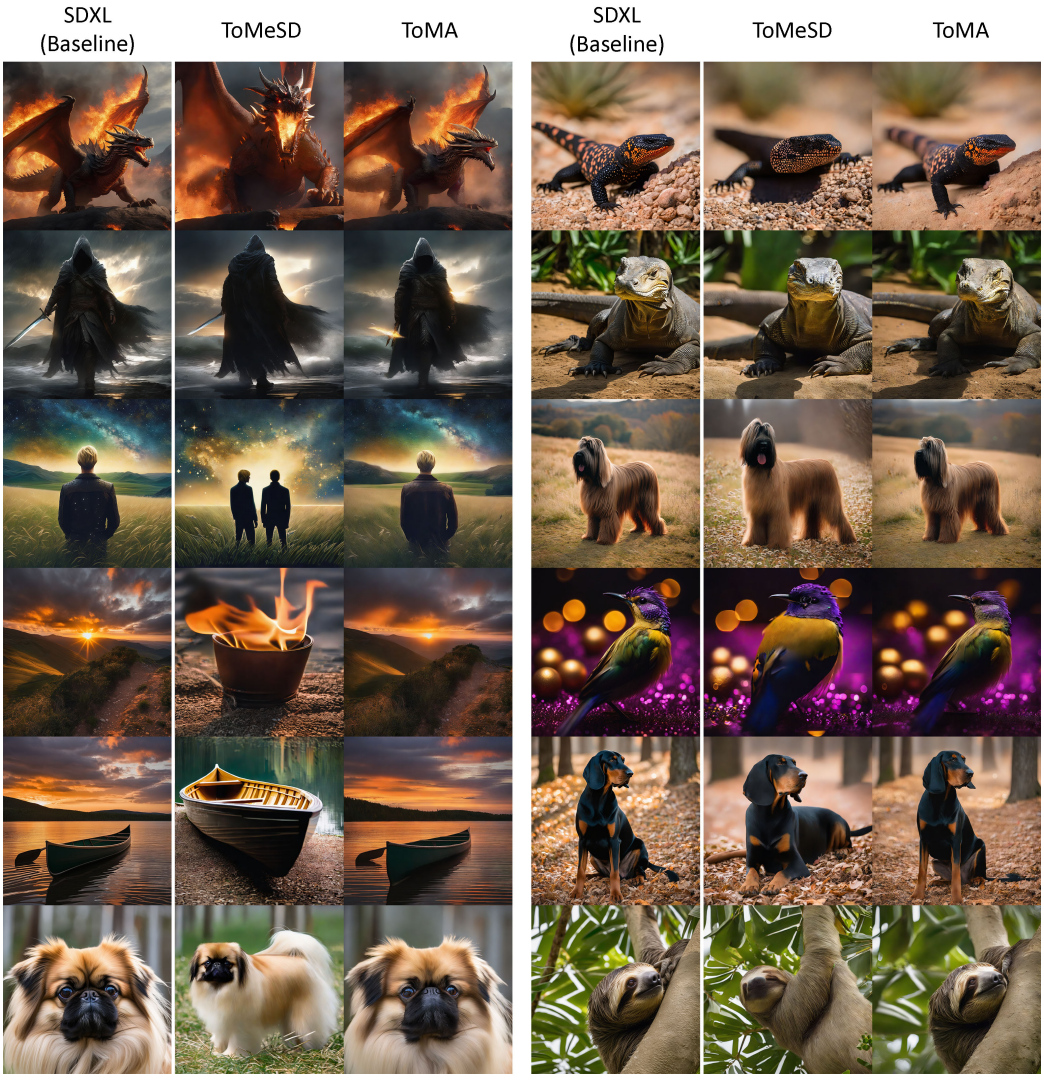


*Figure 6.* Qualitative comparison between Baseline `SDXL-base`, ToMeSD, and ToMA.

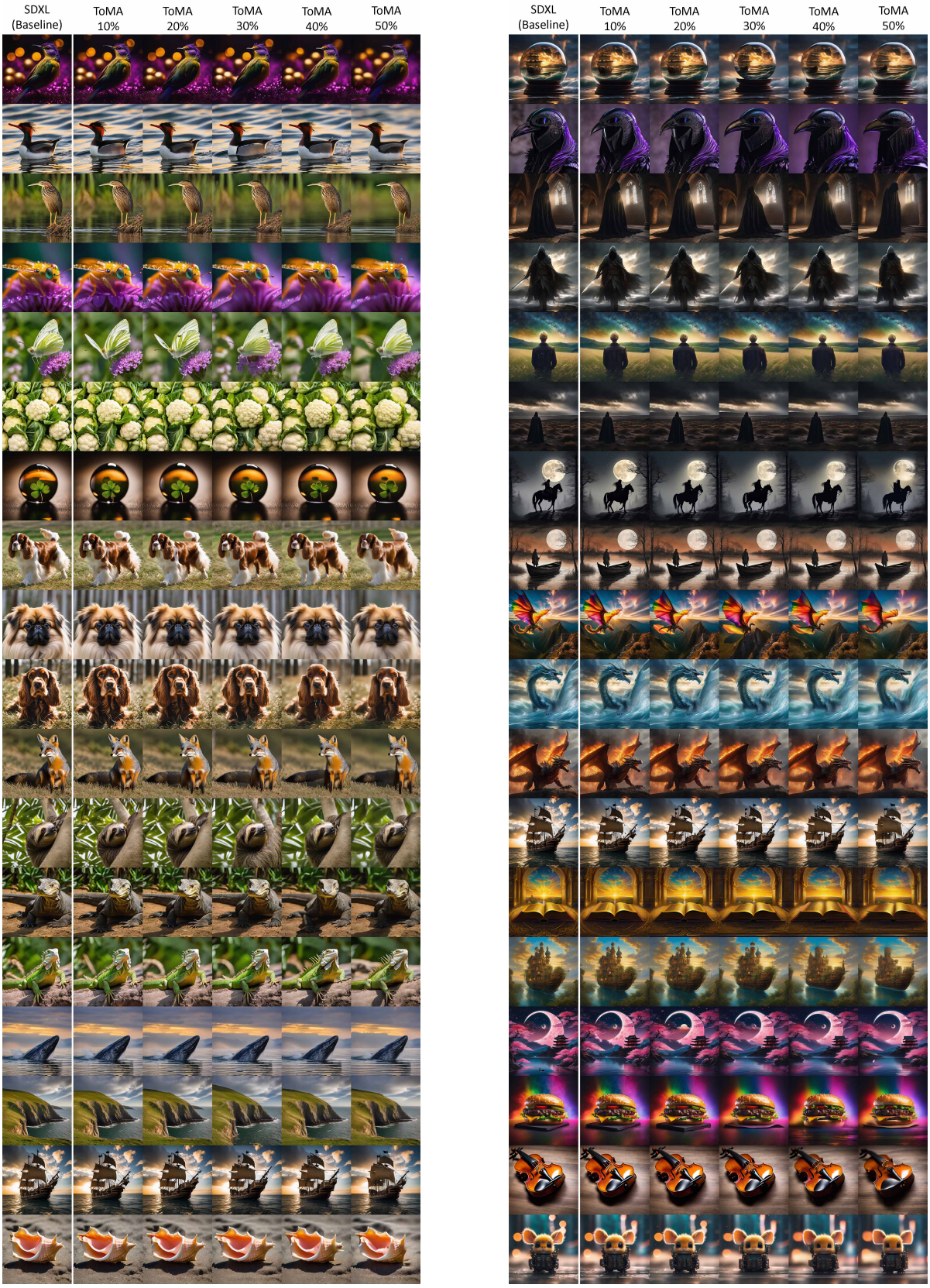


*Figure 7.* **Visual examples of ToMA**. Even with half of the tokens merged, ToMA consistently preserves image quality and often demonstrates greater robustness compared to other methods (ToDo, ToFu, and ToMeSD).

### D.2. Images Generated with Flux

Please refer to Fig. 8 below for more images generated with ToMA on `Flux1.0-dev`.

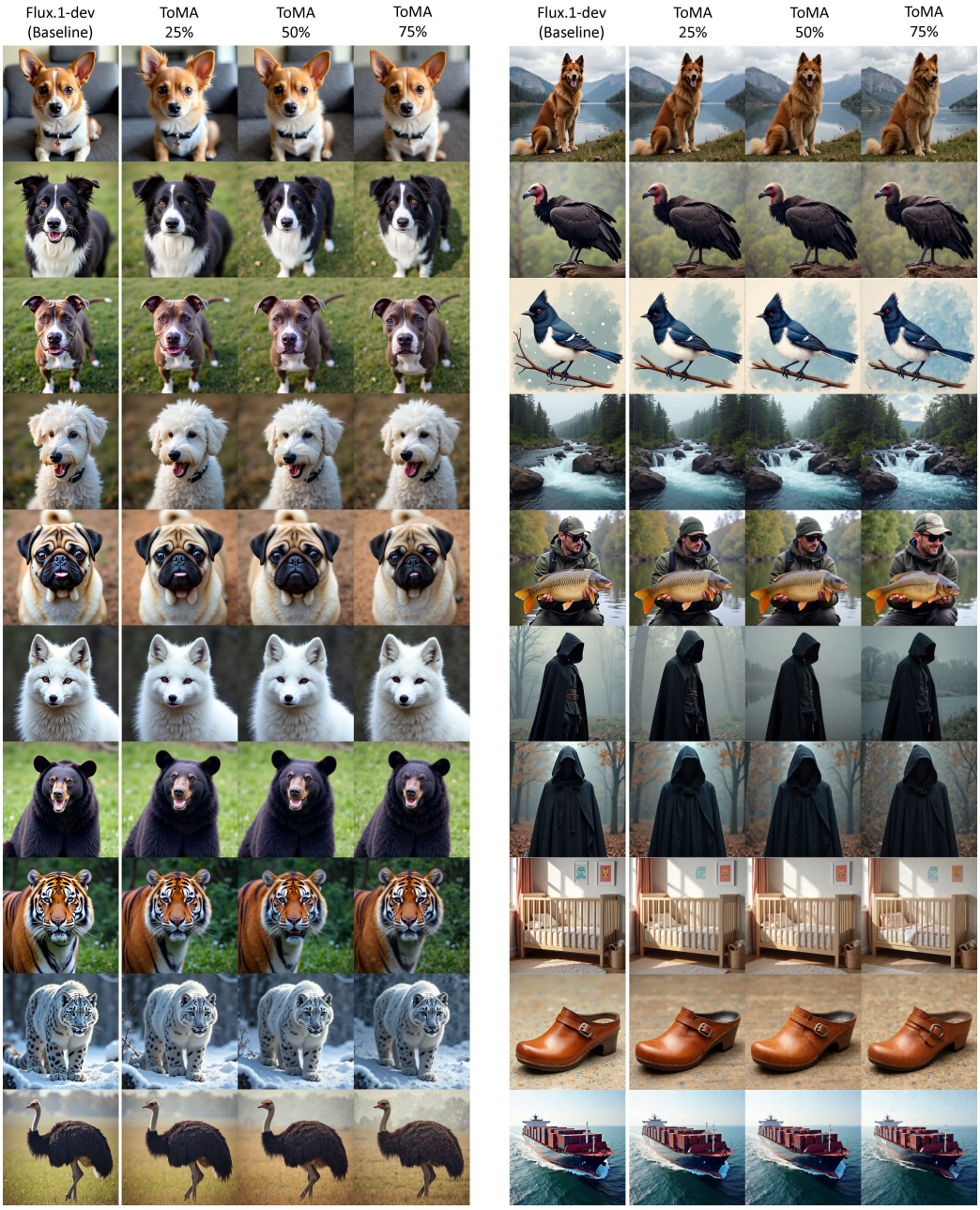


*Figure 8.* Qualitative comparison between Baseline `Flux.1-dev` and ToMA.

# E. Diffusion Transformers (DiT)

## E.1. Locality in DiT

We inspected the hidden states of `Flux`. Using simple visualizations (K-means coloring) at the start of each block and across denoising timesteps, we observed that—even without convolutions—the hidden states already resemble the target image (Fig. 9). This locality is introduced mainly by the rotary embeddings in `Flux`. Empirically, when we apply submodular token selection *within local windows*, the model still produces high-quality images.

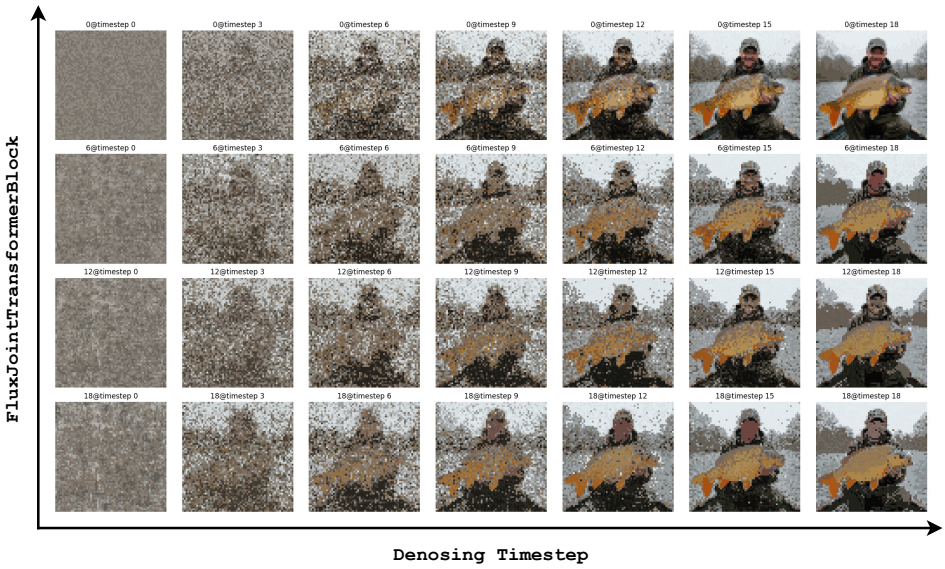


*Figure 9.* Re-colored K-means clusters of hidden states in `Flux.1-dev` across blocks and denoising steps.

## E.2. Transformers and Positional Embeddings in DiT

DiT blocks differ from the usual "self-attention → cross-attention → MLP" pattern, so off-the-shelf merging methods such as ToMeSD, ToFu, or ToDo break the model (all-black / noisy and nonsense outputs). Two issues arise:

1. The DiT block order (attention+MLP fusion) is not aligned with the assumptions in those methods.
2. Positional embeddings (e.g. RoPE) are mixed with both image and text tokens; careless merging discards useful tokens.

We therefore add two simple rules:

- **Skip the first 10 blocks.** Early blocks fuse text and image features; skipping them avoids over-merging.
- **Handle the two DiT block types separately.**

  **JointTransformer.** Text and image tokens are projected *separately*, then concatenated before RoPE. We merge text and image tokens independently, then concatenate; RoPE indices are gathered accordingly.

  **SingleTransformer.** Tokens are already concatenated. We first split the hidden state back into text and image parts, merge each part, then re-concatenate; RoPE is gathered in the same way.

These lightweight changes respect both modality boundaries and positional embeddings, allowing our token-merging variant to run on `Flux` with no perceptible loss in image quality while still delivering the intended speedup.

## F. Ablation Study

All the ablations reported in this appendix are conducted on `SDXL` with a default merge ratio of $r = 0.5$ unless otherwise noted. Each experiment isolates one design choice so that its direct impact on quality—measured by CLIP, DINO, and pixel MSE, and on runtime—measured in seconds per generated image, all can be clearly assessed.

### F.1. Destination–selection strategy

The first experiment evaluates four types of selection windows for choosing destination tokens: a global window that considers every pair of tokens, a local tile facility that performs the same selection inside non-overlapping windows, a horizontal stripe window, and a random baseline on the full window. Tab. 4 shows a clear pattern. Restricting the search to tiles yields the best CLIP and DINO scores and the lowest MSE, while running more than six times faster than the exhaustive global search. The result confirms our locality hypothesis from Fig. 9: most informative tokens lie close to one another, so a global scan is unnecessary and wasteful.

| Type | CLIP ↑ | DINO ↓ | MSE ↓ | Sec/img ↓ |
|---|---|---|---|---|
| Global | 30.949 | 0.069 | 1,637 | 33.2 |
| Tile | 31.019 | 0.055 | 1,274 | 5.1 |
| Stripe | 30.986 | 0.074 | 1,730 | 5.2 |
| Random | 30.553 | 0.090 | 2,029 | 4.5 |

*Table 4.* Comparison of destination–selection rules. Tile-based selection delivers the best quality and the lowest latency. Values are averaged over the SDXL validation prompt set; CLIP/DINO rounded to three decimals, MSE to the nearest integer.

### F.2. Tile granularity

Once the tile–based facility was chosen, we next varied the number of tiles in order to control the spatial extent of each selection window. Intuitively, fewer tiles correspond to larger windows and therefore allow tokens to compete across a wider context, whereas many small tiles enforce highly local competition. Tab. 5 shows four granularities ranging from $4$ large tiles to $256$ small tiles. Moving from $4$ to $16$ tiles yields a large quality jump: DINO improves by $17\%$ and MSE by $14\%$ while latency is cut almost in half. The improvement continues when the window count rises to $64$, which records the best DINO and MSE and also the lowest runtime. At $256$ tiles, DINO and MSE drift upward again, indicating that extremely small windows over-constrain the matching pool; nevertheless, CLIP remains tied with the best value, and the latency does not increase further because the GPU remains compute–bound.

Because the numerical differences between $64$ and $256$ tiles are modest, $64$ tiles strike a cleaner balance: it delivers the strongest quality metrics while preserving the same throughput and avoiding the bookkeeping overhead that arises when thousands of tiny windows must be indexed. For these reasons, we adopt $64$ tiles as the default granularity in the main paper.

| # Tiles | CLIP ↑ | DINO ↓ | MSE ↓ | Sec/img ↓ |
|---|---|---|---|---|
| 4 | 30.775 | 0.069 | 1,564 | 11.4 |
| 16 | 30.991 | 0.057 | 1,345 | 6.4 |
| 64 | 31.019 | 0.055 | 1,274 | 5.0 |
| 256 | 31.027 | 0.057 | 1,296 | 5.0 |

*Table 5.* Influence of tile granularity at a 50% merge ratio. Using 64 tiles achieves the best DINO and MSE while matching the runtime of 256 tiles.

### F.3. Merge and unmerge latency

The third experiment benchmarks the merge and unmerge kernels at a fixed sequence length of $N = 1024$ tokens. We compare the dense linear formulation used in ToMA with the index–scatter implementation in ToMeSD. In ToMeSD the algorithm first builds a destination-index array, then gathers features with `torch.index_select` and finally scatters them back with `index_add_`. Because both gather and scatter operate on the full index list, their cost grows linearly with the merge ratio $r$. Moreover, the discontinuous memory accesses inherent in these calls leave many GPU warps idle.

ToMA eliminates the two passes by replacing them with a single dense matrix multiplication $\tilde{\boldsymbol{A}}\boldsymbol{X}$, where $\tilde{\boldsymbol{A}} \in \mathbb{R}^{D\times N}$ and $D = (1-r)N$. The operation therefore depends only on the output length $D$; its cost is constant with respect to the number of removed tokens and maps efficiently to a single GEMM that fully utilizes GPU compute units. As reported in Table 6, ToMA is consistently four to five times faster than ToMeSD for both merge and unmerge across all tested values of $r$.

| Operation | Method | Time ($\mu$s)↓ | | | Speedup↑ | | |
|---|---|---|---|---|---|---|---|
| | | 25% | 50% | 75% | 25% | 50% | 75% |
| Merge | ToMe | 202.2 | 202.1 | 193.2 | – | – | – |
| | ToMA | 39.0 | 38.8 | 38.8 | 5.2× | 5.2× | 5.0× |
| Unmerge | ToMe | 160.5 | 160.1 | 144.0 | – | – | – |
| | ToMA | 40.2 | 40.5 | 39.6 | 4.0× | 3.9× | 3.6× |

*Table 6.* Micro-benchmarks at sequence length 1024 (median over 1,000 runs on an NVIDIA RTX6000) across different merge ratios. Shaded cells indicate the best result per column. ToMA is roughly four to five times faster than ToMeSD over all merge ratios $r$.

### F.4. Transpose *versus* pseudo–inverse for unmerge

We experiment whether a mathematically exact unmerge, obtained via the Moore–Penrose pseudo–inverse of the merge matrix, offers any quality advantage over the much cheaper transpose. In theory, the pseudo–inverse should restore the pre–merge feature space more faithfully, because it inverts the least–squares projection implicit in the merge. In practice, the merge matrix used by ToMA is highly sparse and close to orthogonal, so the transpose already provides an excellent approximation. Computing the pseudo–inverse requires a QR or SVD decomposition of the $D \times N$ merge matrix, followed by two matrix multiplications to apply the result. These decompositions are memory–bandwidth bound and cannot be fused with the surrounding transformer layers, so the cost is paid in every unmerge step.

Table 7 confirms that the extra work is wasted. Across 300 generated images, the pseudo–inverse gains no measurable improvement: CLIP, DINO, and MSE differ by less than $1\%$. Meanwhile, latency more than doubles because the decomposition incurs additional global synchronizations on the GPU. Given the negligible benefit and the clear timing penalty, we adopt the simple transpose as the default unmerge method.

| Unmerge Method | CLIP ↑ | DINO ↓ | MSE ↓ | Sec/img ↓ |
|---|---|---|---|---|
| Transpose | 31.027 | 0.057 | 1,296 | 4.8 |
| Pseudo–inverse | 30.997 | 0.057 | 1,288 | 10.1 |

*Table 7.* Transpose versus pseudo–inverse unmerge at 50% merge. Quality metrics are identical, but transpose is more than twice as fast.

### F.5. Recompute schedule

The final ablation varies how often destination indices and attention weights are recomputed during denoising. Tab. 8 indicates that refreshing attention every step gives the best overall accuracy, whereas destination indices can be updated ten times less frequently with minimal loss. A schedule of "destination every 10 steps, attention every 5 steps" preserves 99% of the peak quality while roughly halving recompute FLOPs, and is therefore adopted in the main experiments.

| Recompute $D$ | Recompute $\tilde{A}$ | CLIP ↑ | DINO ↓ | MSE ↓ | Sec/img ↓ |
|---|---|---|---|---|---|
| Every 50 steps | Every 50 steps | 30.043 | 0.077 | 2,489 | 4.84 |
| Every 10 steps | Every 10 steps | 30.817 | 0.073 | 1,735 | 4.97 |
| Every 10 steps | Every 5 steps | 30.865 | 0.070 | 1,632 | 5.00 |
| Every 10 steps | Every 1 step | 30.997 | 0.067 | 1,525 | 5.06 |
| Every 5 steps | Every 5 steps | 30.892 | 0.069 | 1,609 | 4.92 |
| Every 1 step | Every 1 step | 30.920 | 0.067 | 1,552 | 5.05 |

*Table 8.* Effect of recomputation frequency at 50% merge. The shaded cells denote the best metric in each column.

## G. Memory analysis

Table 9 provides a peak–memory audit on our methods. For each model, we record both the maximum allocated memory, which reflects the live tensor footprint, and the maximum reserved memory, which includes CUDA's internal caching. Across all three merge ratios the numbers remain tightly clustered: on `Flux`, the largest deviation from the dense baseline is a 0.3% increase in allocated memory for plain ToMA at 25% merging; on `SDXL-base`, the worst case is a 1.9% rise in reserved memory, again for plain ToMA at 25%. The tile variant is even closer and occasionally dips *below* the baseline because smaller activation tensors leave more room for the allocator to reuse blocks.

| Model | Metric | Method | Max Memory (MB)↓ | | |
|---|---|---|---|---|---|
| | | | 25% | 50% | 75% |
| Flux.1-dev | Alloc. | Baseline | 34,640 | 34,640 | 34,640 |
| | | ToMA | 34,744 | 34,710 | 34,675 |
| | | ToMA$_{\text{tile}}$ | 34,647 | 34,647 | 34,642 |
| | Resv. | Baseline | 37,002 | 37,002 | 37,002 |
| | | ToMA | 37,050 | 36,976 | 36,954 |
| | | ToMA$_{\text{tile}}$ | 37,054 | 37,006 | 36,950 |
| SDXL-base | Alloc. | Baseline | 10,721 | 10,721 | 10,721 |
| | | ToMA | 10,931 | 10,857 | 10,797 |
| | | ToMA$_{\text{stripe}}$ | 10,722 | 10,719 | 10,718 |
| | | ToMA$_{\text{tile}}$ | 10,725 | 10,720 | 10,719 |
| | Resv. | Baseline | 14,150 | 14,150 | 14,150 |
| | | ToMA | 14,460 | 14,260 | 14,130 |
| | | ToMA$_{\text{stripe}}$ | 14,114 | 14,188 | 14,222 |
| | | ToMA$_{\text{tile}}$ | 14,158 | 14,158 | 14,182 |

*Table 9.* Peak GPU memory (MB) for different `ToMA` variants on `Flux.1-dev` and `SDXL-base` across three merge ratios. We report both maximum allocated and reserved memory. (↓: lower is better.)

## H. FLOP Analysis

Table 10 reports a layer-wise floating-point-operation (FLOP) breakdown, restricted to the dominant computational modules inside each transformer block: the $QKV$ / output projections and the attention matrix products. Results are shown for the largest block in `Flux` and for the two block types that occur in `SDXL`. Applying ToMA with a 50% merge ratio yields a $2.3\times$ reduction on `Flux` and up to a $3.4\times$ reduction on `SDXL`. The additional FLOPs introduced by ToMA—namely submodular token selection, the merge weight computation, and the linear (un)merge kernels—amount to less than 1 % of the new total and are therefore negligible at the scale of the overall savings.

| Model | Layer Size | FLOPs (G) ↓ | | | Reduction |
|---|---|---|---|---|---|
| | (Seq × Dim) | Original | ToMA (50%) | Overhead | |
| Flux | $4608 \times 3072$ | 520 | 225 | 1.01 | $\sim 2.3\times$ |
| SDXL | $4096 \times 640$ | 106 | 32 | 0.42 | $\sim 3.4\times$ |
| SDXL | $1024 \times 1280$ | 30 | 13 | 0.06 | $\sim 2.4\times$ |

*Table 10.* Layer-level FLOP counts before and after applying ToMA at a 50% merge ratio. The "Overhead" column includes sub-modular selection, merge weight computation, and the linear (un)merge operations.