# GRABME: ACCELERATING VITS FOR SEGMENTATION THROUGH GRAPH-BASED TOKEN MERGING

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Segmentation is a widely studied computer vision task in which Vision Transformers (ViTs) have demonstrated notable performance. Nevertheless, the self-attention mechanism in ViTs incurs quadratic computational complexity with respect to the number of tokens, which has motivated the development of a range of token-merging strategies. Still, existing merging methods based on global matching or rigid local grouping may perturb critical spatial relationships and degrade the preservation of fine-grained details. Furthermore, reliance solely on pairwise similarity metrics may be inadequate for preserving object boundaries and maintaining the integrity of small objects during the merging process. In this work, we introduce Graph-based Token Merging (GrabMe), a training-free token-merging method for efficient segmentation. To more faithfully preserve relative spatial structure, GrabMe constrains merging to one-hop neighbors on a spatial adjacency graph, where resulting token groups inherit adjacencies of their constituent tokens. For parallel merging, GrabMe employs dynamic programming to construct a conflict-free pool of candidates with maximum cardinality. Subsequently, complementary structural context derived from one-hop similarity distributions is incorporated into the merge decision, thereby mitigating the loss of fine-grained details. Extensive experiments with ViT backbones across semantic, instance, and panoptic segmentation benchmarks demonstrate that GrabMe effectively improves the accuracy–efficiency trade-off of pretrained backbones for segmentation without requiring any additional training. Our code is available at this link.

## 1 INTRODUCTION

Segmentation tasks require models to make predictions at each spatial location in an image, where preserving relative spatial structure and fine-grained details such as object boundaries and small objects is essential. Consequently, segmentation pipelines typically process high-resolution inputs to retain sufficient spatial information. Vision Transformers (ViTs; Dosovitskiy et al., 2021) have achieved notable performance across a wide range of segmentation tasks due to their ability to model rich global context through self-attention. However, self-attention is repeatedly applied over token sequences, with computational complexity scaling quadratically in the number of tokens, leading to significant inefficiency when processing high-resolution inputs. Accordingly, a range of token reduction methods, including token merging, have been proposed in an effort to enhance the efficiency of pretrained ViTs while preserving their strong performance, particularly for segmentation.

As a representative token-merging method, ToMe (Bolya et al., 2023) employed an efficient bipartite soft-matching pipeline to merge redundant tokens within transformer blocks. Nevertheless, its global matching strategy without spatial constraints may distort the relative spatial structure that is essential for accurate segmentation. Meanwhile, several approaches have explored spatially constrained merging strategies (Lu et al., 2023; Norouzi et al., 2024; Liang et al., 2022; Haroun et al., 2024; 2025). Still, fine-grained local merging can incur additional computational overhead relative to simple global matching, particularly when it relies on iterative clustering or pairwise merging (Haurum et al., 2024; Haroun et al., 2025). Restricting local merging to a few predefined layers (Liang et al., 2022; Li et al., 2023) reduces this overhead while hindering the use of representations refined in subsequent layers for merge decisions. To simplify local matching operation itself, grouping schemes such as fixed windows or predefined adjacency (Norouzi et al., 2024; Haroun et al., 2024) have also been explored, but afford limited flexibility in reflecting complex boundary structures.



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.

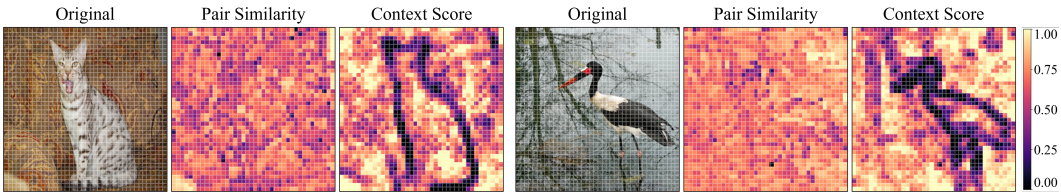



Figure 1: **Visualization of context scores in GrabMe.** Each example presents the original image alongside maps of each token's similarity to its best-matching adjacent neighbor and its context score, with lower context scores near object boundaries or within small, thin structures.

Furthermore, prior local merging approaches still rely solely on the similarity among the tokens within each merge candidate. As illustrated in Figure 1, we observed that tokens in sensitive regions, such as near object boundaries or within small, thin structures, may form pairs with their most similar neighbors and still exhibit high pairwise similarity, making such cases difficult to identify from pairwise similarity alone. By contrast, when similarities to other adjacent tokens alongside the within-pair similarity are collectively treated as a *distribution*, the corresponding mean and standard deviation make pairs in sensitive regions clearly separable from those in homogeneous regions. These patterns suggest that local similarity distributions can offer structural context for assessing merge stability and preserving fine-grained details in segmentation.

Thus motivated, we propose Graph-based Token Merging (GrabMe), a training-free token-merging framework for high-resolution segmentation that leverages dynamic graph connectivity. GrabMe constructs a spatial adjacency graph over the initial patch tokens and progressively inherits its connectivity as tokens are merged. At each layer, GrabMe restricts feature similarity computation to one-hop neighbors and identifies each token's most similar neighbor as a potential merge target. To enable parallel merging, GrabMe builds a candidate pool through dynamic programming (Chen et al., 1988), maximizing the number of feasible candidates while ensuring that no merge target is itself simultaneously merged into another token. Subsequently, the *context score*—computed from the mean and standard deviation of each token's one-hop similarity distribution—is integrated with the pairwise similarity measure to determine the final merge decision. By progressively expanding graph connectivity, GrabMe enables local merging that adapts to irregular spatial configurations, while the context score regulates merge operations to preserve fine-grained structural details.

We evaluate GrabMe across various segmentation tasks, including semantic, instance and panoptic segmentation with ViT backbones. Under matched task-level performance, GrabMe consistently delivers higher throughput than other token-merging baselines through more aggressive token reduction. Statistical analyses further provide corroborating evidence that, compared with other baselines, GrabMe constructs token groups with higher semantic consistency while more reliably preserving spatially sensitive tokens near object boundaries and within small objects. Taken together, these results demonstrate that GrabMe substantially accelerates inference while maintaining superior performance across a diverse set of segmentation tasks, without necessitating any additional training.

## 2 Related Work

**Segmentation.** Segmentation requires fine-grained predictions across image locations. Early approaches to semantic segmentation predominantly employed convolutional neural networks (CNNs) (Badrinarayanan et al., 2017; Chen et al., 2018; Long et al., 2015; Zhao et al., 2017), whereas more recent studies have investigated architectures based on ViTs (Wang et al., 2021; Xie et al., 2021; Zhang et al., 2022; Zheng et al., 2021; Fang et al., 2023). Segmenter (Strudel et al., 2021) demonstrates the effectiveness of plain Transformer encoders for semantic segmentation. Subsequent approaches, including Mask2Former (Cheng et al., 2022), ViT-Adapter (Chen et al., 2023b), and EoMT (Kerssies et al., 2025), support semantic, instance, and panoptic segmentation within unified architectures. However, the computational complexity of self-attention scales quadratically with the number of tokens, posing a significant bottleneck for segmentation at high resolutions.

**Token Reduction.** The computational inefficiency of ViTs has prompted the development of a range of token-reduction techniques. Token Pruning methods, including DynamicViT (Rao et al.,

2021) and subsequent approaches (Bergner et al., 2025; Chen et al., 2023a; Fayyaz et al., 2022; Kong et al., 2022; Tang et al., 2023; Yin et al., 2022), discard tokens based on estimated importance, often relying on auxiliary mechanisms or training to mitigate the consequent information loss.

Alternatively, token merging reduces the number of tokens by aggregating redundant tokens rather than discarding them (Chen et al., 2023a; Feng & Zhang, 2023; Kim et al., 2024; Wang & Yang, 2025; Zhang et al., 2025). ToMe (Bolya et al., 2023) introduces a bipartite soft matching pipeline, while subsequent methods refine merging through importance estimation, alternative similarity measures, or clustering strategies (Tran et al., 2024; Lee et al., 2024; Haurum et al., 2024). However, merging without spatial constraints may disrupt relative spatial structures crucial for segmentation.

Meanwhile, studies on local merging explore incorporating spatial constraints into token merging. CTS (Lu et al., 2023) and ALGM (Norouzi et al., 2024) use simple fixed windows, limiting their flexibility around complex boundaries. Expedite (Liang et al., 2022) adopts K-means clustering but may not adequately capture varying information redundancy, whereas AiluRus (Li et al., 2023) selects representative tokens through density-based clustering. Recent works directly exploit token adjacency. LoTM (Haroun et al., 2024) employs predefined, disjoint horizontal pairs. DHTM (Haroun et al., 2025) considers broader scope of neighborhood but merges pairs sequentially, limiting parallel operation. G2TM (Bercy et al., 2026) performs threshold-based merging on a spatial adjacency graph, while merging is restricted to a single early layer. Overall, an efficient, progressive local merging strategy that more faithfully preserves spatial structures warrants further investigation.

## 3 Proposed Method

GrabMe progressively reduces tokens through graph-based token merging, as illustrated in Figure 2 and detailed in Section 3.1. At each layer, tokens identify their most similar one-hop neighbors as merge targets, from which GrabMe builds the largest conflict-free candidate pool for parallel merging via dynamic programming (Section 3.2). Candidate pairs are then ranked by pair similarity, in conjunction with a context score derived from one-hop similarity distributions, in order to prevent the merging of tokens located in sensitive regions (Section 3.3). The highest-ranked pairs are then selected for merging and resulting token groups inherit the one-hop adjacencies of their constituent tokens. After the final merge layer, GrabMe restores the original sequence length for decoding.

### 3.1 Token Merging on a Spatial Adjacency Graph

GrabMe initializes a four-connected adjacency graph over the initial patch tokens, connecting horizontal and vertical neighbors. At any merge layer, each token $u$ represents a token group formed by preceding merges, with representation $\mathbf{x}_u$ and size $m_u$ denoting the number of original patches contained in the group. We further maintain a group identifier $g_i$ for each original patch $i$, indicating its current token group. Initially, each token forms a singleton group containing itself, such that $m_u = 1$ and $g_i = i$. We describe a single merge layer below and omit layer indices for clarity.

Given the current graph, each token identifies its most similar one-hop neighbor as a potential merge target. GrabMe then constructs a conflict-free candidate pool, ensuring that any token selected as a merge target is not simultaneously merged into another token, thereby enabling parallel merging without pairwise conflicts. The candidate pairs are subsequently ranked using pairwise similarity together with the context score. The top-$k$ candidate pairs are then selected for merging, where $k$ denotes the merge budget and specifies the number of tokens merged at each layer. A larger $k$ means that more tokens are merged at each layer, resulting in a higher reduction rate. For each merge target $v$, let $\mathcal{A}_v$ contain $v$ itself and the tokens that select $v$ as their target. We aggregate their features by:

$$\mathbf{x}'_v = \frac{\sum_{u\in\mathcal{A}_v} m_u \mathbf{x}_u}{\sum_{u\in\mathcal{A}_v} m_u}, \qquad m'_v = \sum_{u\in\mathcal{A}_v} m_u. \tag{1}$$

Each constituent token contributes according to its size through weighted averaging, while the sequence length is reduced from $N$ to $N-k$ for every sample in the batch. Then, each resulting group inherits the adjacencies of its constituent tokens, while intra-group adjacencies are removed and inter-group adjacencies are retained, thereby gradually expanding graph connectivity across layers.

The group identifier of each original patch is also updated to indicate its current group, and the compact token sequence continues through the encoder . However, for segmentation, the decoder

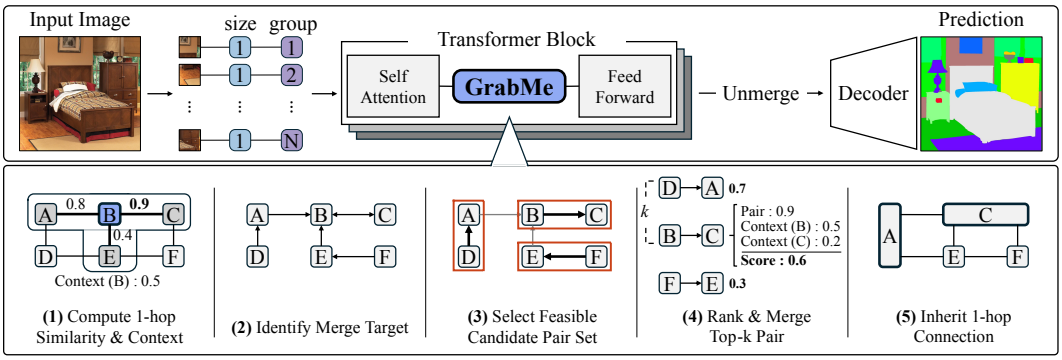



Figure 2: **Overview of GrabMe.** GrabMe progressively merges tokens within Transformer blocks and restores the dense token sequence before decoding. Each merge layer consists of five steps: (1) compute one-hop feature similarities and context scores; (2) identify each token's most similar neighbor as its merge target; (3) construct a maximum-cardinality conflict-free candidate pool; (4) rank candidate pairs based on pairwise similarity and the context scores, then merge the top-$k$ pairs in parallel; and (5) update graph connectivity by inheriting the adjacencies of constituent tokens.

commonly requires a token sequence aligned with the original patch grid. To this end, GrabMe restores the dense sequence through a single gather operation using the group identifiers to map compact tokens back to their original patch positions, an operation we refer to as *unmerge*.

### 3.2 Conflict-Free Candidate Pair Selection

This section details how GrabMe builds a conflict-free candidate pool for parallel merging. For each token $u$, we compute the cosine similarity $s_{uv} = \cos(\mathbf{x}_u, \mathbf{x}_v)$ between its representation and those of its one-hop neighbors $v \in \mathcal{N}(u)$. Subsequently, we define the target $d(u)$ as the neighbor that achieves the maximal similarity and construct the corresponding pair as directed edge $(u, d(u))$:

$$d(u) = \arg\max_{v \in \mathcal{N}(u)} s_{uv}. \tag{2}$$

Yet not all pairs are eligible for parallel merging: a conflict arises when a merge target also selects another token as its own merge target. Such conflicts render merge operations order-dependent and preclude an exact reduction of $k$ tokens. Therefore, we construct a candidate set in which no merge target is itself merged into another token, while allowing multiple tokens to share the same target.

As each ordered pair $(u, d(u))$ defines a directed edge, the resulting graph decomposes into cycles with attached trees, allowing GrabMe to apply dynamic programming (DP; Chen et al. (1988)) to obtain an exact maximum-cardinality conflict-free candidate pool. For an attached tree, we compute the maximum number of valid merge candidates under two cases for each token $u$, denoted by $D_u^1$ and $D_u^0$ when $u$ is selected and excluded, respectively. For a leaf token, defined as a token that no other token selects as its merge target, these values are initialized as $D_u^1 = 1$ and $D_u^0 = 0$. Moving upward from the leaves, we recursively update the values over tokens $v$ satisfying $d(v) = u$ as

$$D_u^1 = 1 + \sum_{v:\, d(v)=u} D_v^0, \qquad D_u^0 = \sum_{v:\, d(v)=u} \max\left\{D_v^0, D_v^1\right\}. \tag{3}$$

When $u$ is selected, every $v$ must be excluded to satisfy the conflict constraint; when $u$ is excluded, each $v$ can independently take its better state. The results obtained for the trees are then incorporated into the tokens on the cycle. For a cycle token $p$ and non-cycle tokens $v$ where $d(v) = p$, we define

$$C_p^1 = 1 + \sum_{v:\, d(v)=p} D_v^0, \qquad C_p^0 = \sum_{v:\, d(v)=p} \max\left\{D_v^0, D_v^1\right\}. \tag{4}$$

For each of the $K$ disjoint cycles, we apply the same recurrence using the $C_p^1$ and $C_p^0$ values. As a cycle has no leaf token, we take an arbitrary token on the cycle as a reference and evaluate the entire cycle under both inclusion and exclusion of the reference token. The better of the two cases

defines the optimal candidate set $\mathcal{C}_k^*$ for the $k$-th cycle and its incoming trees. The final conflict-free candidate set $\mathcal{S}^*$ is constructed by aggregating the optimal selections identified across all cycles.

Each token is visited only a constant number of times, yielding an overall selection complexity of $O(N)$. The final $k$ pairs are then selected from $\mathcal{S}^*$ based on pairwise similarity and context score. In effect, this DP-based formulation enables efficient parallel merging by constructing a conflict-free candidate pool while providing as many feasible options as possible for the final merge decision.

### 3.3 Context-Aware Merging

Pair similarity alone may be insufficient to guarantee that a merging procedure preserves fine-grained structural properties. Tokens situated along object boundaries or thin, small-scale structures may display elevated similarity; however, aggregating tokens in these regions can lead to substantial information loss, which is particularly detrimental for segmentation. In this regard, reliable merge decisions should consider similarity distributions around the endpoints beyond pair similarity alone.

For each token $u$, let $\mu_u$ and $\sigma_u$ denote the mean and standard deviation of its feature similarities to one-hop neighbors, rescaled to [0,1]. We then define the context score of $u$ as

$$c_u = \mu_u - \sigma_u, \text{ where } \mu_u \in [0,1],\ \sigma_u \geq 0. \tag{5}$$

The standard deviation penalizes heterogeneous regions, where varying similarities to one-hop neighbors may indicate a semantic boundary. Since low variance alone can arise when a token is uniformly dissimilar to all of its neighbors, $\mu_u$ complements this penalty. Consequently, $c_u$ favors tokens that are consistently similar to both their merge targets and other neighboring tokens.

For a feasible candidate pair $(u, v)$ in $\mathcal{S}^*$, we combine pairwise similarity and the context scores as

$$R_{u,v} = \frac{1}{2}\left(q_{uv} + \frac{c_u + c_v}{2}\right), \tag{6}$$

where $q_{uv}$ denotes the rescaled cosine similarity between $u$ and $v$. The top-$k$ candidate pairs ranked by $R_{u,v}$ are selected for merging, where $k$ denotes the per-layer merge budget. In effect, by capturing region-level patterns beyond pair similarity, the context score preferentially keeps tokens in sensitive regions from being merged, effectively preserving information essential for segmentation.

## 4 Experiments

**Experimental setup.** We evaluate GrabMe across semantic, instance, and panoptic segmentation. Task-specific benchmarks include ADE20K (Zhou et al., 2017), Cityscapes (Cordts et al., 2016), COCO-Stuff (Caesar et al., 2018) and COCO (Lin et al., 2014). For these evaluations, we use Segmenter-Large, EoMT, and EVA with ViT-Adapter. Token-merging baselines include ToMe, PiToMe, MCTF, ATC, ALGM, Expedite, LoTM, DHTM, and G2TM. We report task-specific evaluation metrics—mean Intersection over Union (mIoU), mask Average Precision (AP), and Panoptic Quality (PQ)—to quantify segmentation performance, together with computational efficiency measured as throughput in images per second (im/s). Methods are evaluated either (1) at their respective maximum token reduction rates that maintain backbone performance or (2) under a common high reduction-rate setting. Reduction rates are presented in the tables as percentage rates (%). All experiments are conducted on a single NVIDIA RTX A6000 without any additional training.

### 4.1 Comparison on Semantic Segmentation

Table 1 presents the results on semantic segmentation with Segmenter-Large (Strudel et al., 2021) across ADE20K, Cityscapes, and COCO-Stuff. Each baseline is evaluated at the highest token-reduction rate that maintains the backbone mIoU; otherwise, we report the highest mIoU achieved and the corresponding token-reduction rate. For instance, ALGM and Expedite attain peak mIoU scores of 78.8 and 78.9, respectively on Cityscapes, which remain below the backbone performance of 79.1; the token-reduction rate and throughput could only be quantified under these controls.

On ADE20K, GrabMe achieves approximately 21.6 percentage points greater token reduction than the strongest baseline, while improving throughput by more than 70% over the backbone. These advantages are likewise evident on Cityscapes and COCO-Stuff, where throughput improves by more

Table 1: **Results on the validation sets of semantic segmentation benchmarks.** Each method is evaluated at the maximum token reduction rate preserving the backbone mIoU; otherwise, we report the closest achieved mIoU and corresponding reduction rate, with mIoU highlighted in red. All baselines are applied without additional training. Throughput (im/s) is measured with batch size 32.

| Method | ADE20K | | | Cityscapes | | | COCO-Stuff | | |
|---|---|---|---|---|---|---|---|---|---|
| | mIoU | rate (%) | im/s (↑) | mIoU | rate (%) | im/s (↑) | mIoU | rate (%) | im/s (↑) |
| Segmenter-Large | 51.8 | 0.0 | 9.0 | 79.1 | 0.0 | 5.6 | 45.6 | 0.0 | 8.9 |
| *Global-range merging* | | | | | | | | | |
| + ToMe ICLR'23 | 51.8 | 39.0 | 9.9 | 79.1 | 39.6 | 6.2 | 45.6 | 4.5 | 7.5 |
| + PiToMe NeurIPS'24 | 51.8 | 10.5 | 9.1 | 79.1 | 10.4 | 5.7 | 45.6 | 70.5 | 13.6 |
| + MCTF CVPR'24 | 51.8 | 40.2 | 7.1 | 79.1 | 39.9 | 4.2 | 45.6 | 70.4 | 9.1 |
| + ATC ECCV'24 | 51.8 | 42.8 | 1.5 | 79.1 | 29.7 | 0.5 | 45.6 | 26.3 | 1.1 |
| *Local-range merging* | | | | | | | | | |
| + G2TM VISAPP'26 | 51.8 | 16.8 | 7.7 | 79.1 | 2.4 | 4.6 | 45.6 | 4.5 | 7.6 |
| + DHTM VISAPP'25 | 51.8 | 20.0 | 1.9 | 79.1 | 10.0 | 1.5 | 45.6 | 45.9 | 1.2 |
| + LoTM ICONIP'24 | 51.8 | 30.0 | 10.4 | 79.1 | 13.0 | 6.0 | 45.5 | 0.9 | 8.9 |
| + ALGM CVPR'24 | 51.8 | 31.0 | 9.6 | 78.8 | 23.0 | 6.7 | 45.6 | 40.4 | 9.9 |
| + Expedite NeurIPS'22 | 51.8 | 47.4 | 13.0 | 78.9 | 8.2 | 5.6 | 45.6 | 60.9 | 14.9 |
| + GrabMe w/o Context | 51.8 | 48.0 | 12.9 | 79.1 | 13.5 | 6.1 | 45.6 | 61.5 | 14.2 |
| + GrabMe (Ours) | 51.8 | **69.0** | **15.6** | 79.1 | **44.8** | **8.0** | 45.6 | **72.0** | **15.7** |

Table 2: **Results on ADE20K with recent ViT-based segmentation backbones.** Throughput is measured with batch size 32 for EoMT, and 8 for EVA due to memory constraints.

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| EoMT-Large | 59.5 | 0.0 | 15.6 |
| + ToMe | 59.5 | 2.0 | 13.4 |
| + PiToMe | 59.5 | 3.9 | 12.2 |
| + G2TM | 57.5 | 30.7* | 14.1 |
| + LoTM | 59.5 | 10.0 | 13.0 |
| + ALGM | 59.5 | 7.4 | 19.6 |
| + Expedite | 57.9 | 39.0* | 16.3 |
| + GrabMe (Ours) | 59.5 | **37.1** | **21.8** |

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| EVA & ViT-Adapter | 61.5 | 0.0 | 1.7 |
| + ToMe | 61.5 | 39.5 | 1.9 |
| + PiToMe | 61.5 | 10.2 | 1.6 |
| + G2TM | 61.5 | 19.4 | 1.8 |
| + LoTM | 61.5 | 75.0 | 2.2 |
| + ALGM | 61.6 | 66.8 | 2.2 |
| + Expedite | 61.5 | 41.0 | 1.1 |
| + GrabMe (Ours) | 61.5 | **75.3** | **2.5** |

* The reported reduction rate is not directly comparable.

than 40% and 75%, respectively, without degrading the backbone mIoU. These results indicate that the effectiveness of GrabMe generalizes across datasets with diverse scenes and class distributions.

The strengths of GrabMe are not confined to the Segmenter architecture but also extend to recent ViT-based segmentation backbones. As shown in Table 2, GrabMe improves EoMT throughput by about 40% without degrading mIoU. In contrast, G2TM and Expedite substantially reduce mIoU, rendering their reported reduction rates not directly comparable. On EVA with ViT-Adapter, GrabMe reduces tokens by over 75% while improving throughput by about 47%. These results suggest that efficiency improvements of GrabMe are not limited to a particular backbone architecture.

### 4.2 Comparison on Instance, Panoptic Segmentation

Table 3 presents the results on instance and panoptic segmentation with EoMT on COCO. Complementing the preceding evaluation of efficiency at preserved accuracy, we compare task performance at a matched token reduction rate of approximately 80%. This setting assesses how well each method retains segmentation quality under aggressive token merging, measured by mask AP and PQ.

In this comparison, local-range baselines tend to exhibit larger performance degradation than their global-range counterparts, while GrabMe retains the highest task performance among all token-merging baselines. For instance segmentation, GrabMe limits the relative decrease in mask AP to approximately 7% while increasing throughput by approximately 54% over the original backbone. Notably, GrabMe achieves the highest $AP_S$ among the merging methods, retaining performance on small objects even as the other local-range baselines exhibit substantial degradation. This result

Table 3: **Results on instance and panoptic segmentation on COCO.** All baselines are applied without additional training. We report mask AP and PQ for instance and panoptic segmentation, respectively. Throughput (im/s) is measured with batch size 32.

| Method | Instance Segmentation | | | | | Panoptic Segmentation | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $AP_{mask}$ | $AP_{75}$ | $AP_S$ | rate (%) | im/s (↑) | PQ | $PQ^{Th}$ | $PQ^{St}$ | rate (%) | im/s (↑) |
| EoMT-Large | 44.4 | 47.0 | 19.7 | 0.0 | 7.9 | 56.0 | 61.2 | 48.2 | 0.0 | 8.0 |
| + ToMe | 40.4 | 42.0 | 16.2 | 80.0 | 8.8 | 53.3 | 57.8 | 46.4 | 80.0 | 8.9 |
| + PiToMe | 39.9 | 41.5 | 17.9 | 80.0 | 9.7 | 52.3 | 57.2 | 45.0 | 80.0 | 9.8 |
| + G2TM | 16.6 | 15.1 | 9.3 | 79.9 | 10.6 | 25.0 | 30.5 | 16.8 | 79.7 | 10.8 |
| + LoTM | 34.4 | 35.5 | 7.6 | 80.0 | 9.0 | 50.1 | 52.3 | 46.8 | 80.0 | 9.1 |
| + ALGM | 32.7 | 32.8 | 12.6 | 79.8 | 12.0 | 46.8 | 51.6 | 39.6 | 79.7 | 12.0 |
| + Expedite | 28.4 | 28.5 | 1.8 | 79.8 | 8.6 | 45.2 | 45.4 | 45.0 | 79.8 | 8.7 |
| + GrabMe (Ours) | 41.3 | 43.4 | 18.3 | 80.0 | **12.2** | 53.8 | 58.4 | 46.8 | 80.0 | **12.3** |

Table 4: **Ablation studies** on ADE20K with Segmenter-Large, evaluating the effects of the context score and candidate selection strategy. For each ablation, we report mIoU at a fixed token reduction rate and, conversely, the maximum reduction rate attainable while matching the best mIoU.

(a) **Context score**.

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| Pair sim. | 51.1 | 69.0 | **15.6** |
| Context | 51.2 | 69.0 | **15.6** |
| Pair sim. & Context | **51.8** | 69.0 | **15.6** |

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| Pair sim. | 51.8 | 48.0 | 12.9 |
| Context | 51.8 | 61.5 | 14.5 |
| Pair sim. & Context | 51.8 | **69.0** | **15.6** |

(b) **Candidate selection**.

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| Greedy Selection | 51.3 | 69.0 | 13.2 |
| Bipartite Partition | 51.5 | 69.0 | **15.6** |
| DP Selection (Ours) | **51.8** | 69.0 | **15.6** |

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| Greedy Selection | 51.8 | 49.5 | 11.0 |
| Bipartite Partition | 51.8 | 57.0 | 14.0 |
| DP Selection (Ours) | 51.8 | **69.0** | **15.6** |

aligns with the established role of context-aware merging in preserving tokens from sensitive regions such as small objects; we further investigate this effect in Section 4.4.

A similar trend holds for panoptic segmentation: GrabMe achieves the highest PQ among the baselines, while improving throughput by more than 50% over the backbone. These results demonstrate that the effectiveness of GrabMe extends to instance and panoptic segmentation, offering stronger performance retention under aggressive token reduction with substantial inference speedups.

### 4.3 Ablation Studies

**Context Score.** Table 4a evaluates the contribution of the context score. At a fixed reduction rate, the context score alone outperforms pair similarity, while combining both yields the best mIoU without additional overhead. These results support the view that the context score effectively complements pair similarity. This benefit becomes more pronounced under matched mIoU: relying on pair similarity alone lowers the token reduction rate by 21.0 percentage points, with throughput correspondingly decreasing from 15.6 to 12.9 im/s. These results highlight the context score as a key factor enabling GrabMe to sustain higher token reduction than other baselines at matched mIoU.

**Candidate Pair Selection Strategy.** We next ablate the selection strategy used to construct the candidate set. Table 4b compares our DP-based selection with greedy selection, which sequentially selects high-similarity pairs, and bipartite partitioning of ToMe, which divides tokens into source and destination sets and restricts matching to source-to-destination pairs. At a fixed reduction rate, greedy selection decreases both throughput and mIoU, while bipartite partitioning reduces mIoU. Under matched mIoU, greedy selection and bipartite partitioning operate at reduction rates 19.5 and 12.0 percentage points lower, respectively. This advantage stems from DP-based selection maximizing candidate-pool cardinality under the conflict constraint, providing more candidate pairs for the merge decision without additional runtime overhead relative to simple bipartite partitioning.

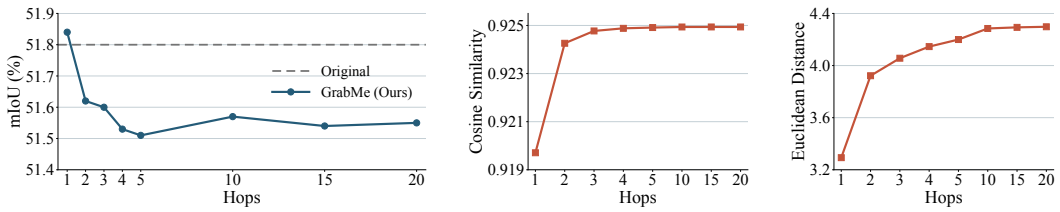



Figure 3: **Effect of the one-hop constraint on ADE20K.** We report mIoU, average feature similarity of merged pairs and spatial displacement from original token positions as the hop limit increases.

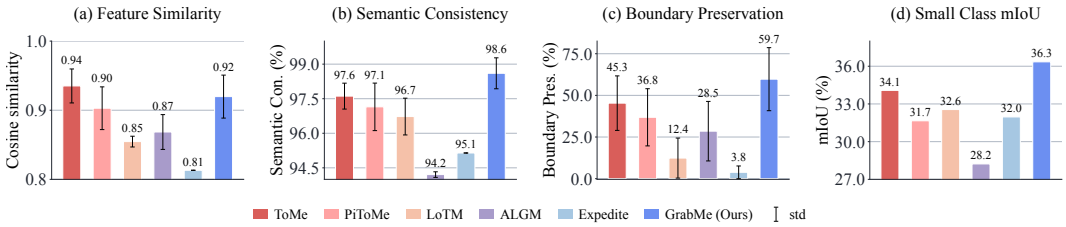



Figure 4: **Statistical properties of token-merging behavior.** We compare (a) the average feature similarity of merged pairs, (b) the semantic consistency of resulting token groups, (c) the preservation of boundary tokens, and (d) the mIoU over the 10 classes with the smallest ground-truth areas.

**Effect of Spatial Constraint.** We further examine the spatial constraint of GrabMe by increasing the hop limit, allowing each token to merge over a broader range. As shown in Figure 3, mIoU gradually decreases as the search space expands despite increasing pair similarity, while merged tokens exhibit greater spatial displacement, indicating that the one-hop constraint keeps merged tokens closer to their original positions. These results underscore the primacy of spatial structure preservation for segmentation and substantiate the effectiveness of the one-hop constraint in GrabMe.

### 4.4 Merging Behavior Analysis and Qualitative Comparison

To systematically characterize differences in token-merging behavior, we compare GrabMe with baselines through statistical and qualitative analyses. All comparisons use Segmenter-Large at the same token reduction rate; statistical properties are measured on ADE20K and summarized in Fig. 4, while qualitative merging behavior is visualized in Fig. 5.

**Statistical Analysis.** As shown in Figure 4(a) and (b), GrabMe selects pairs with lower feature similarity than ToMe due to its spatially constrained search space, while the resulting token groups achieve the highest semantic consistency among all methods, measured by the dominant-class proportion per merged group. At the same time, Figure 4(c) demonstrates that GrabMe more reliably keeps boundary tokens–tokens located along region boundaries in the ground-truth–from being merged. This behavior is consistent with the role of the context score and such selective preservation is particularly consequential for classes with small regions. To probe this effect, we compare the mIoU over the 10 classes with the smallest ground-truth areas in Figure 4(d), where GrabMe achieves the highest mIoU among all methods. Together, these results support the view that GrabMe forms semantically coherent token groups while reliably preserving spatially sensitive tokens.

**Qualitative Analysis.** We qualitatively compare merging behavior at a matched reduction rate in Figure 5. ToMe and PiToMe perform global merging, producing spatially discontiguous groups that may potentially distort relative spatial structure. G2TM incorporates graph adjacency but merges at a single early layer, which may lead to abrupt aggregation based on low-level representations. Expedite is likewise restricted to single-layer merging and does not account for varying information density. LoTM and ALGM rely on rigid local grouping schemes that are less flexible for complex region shapes. In contrast, GrabMe effectively adapts to irregular region shapes through graph connectivity and leverages gradually refined representations across layers. The context score further discourages merging in sensitive regions, enabling GrabMe to better preserve fine-grained details.

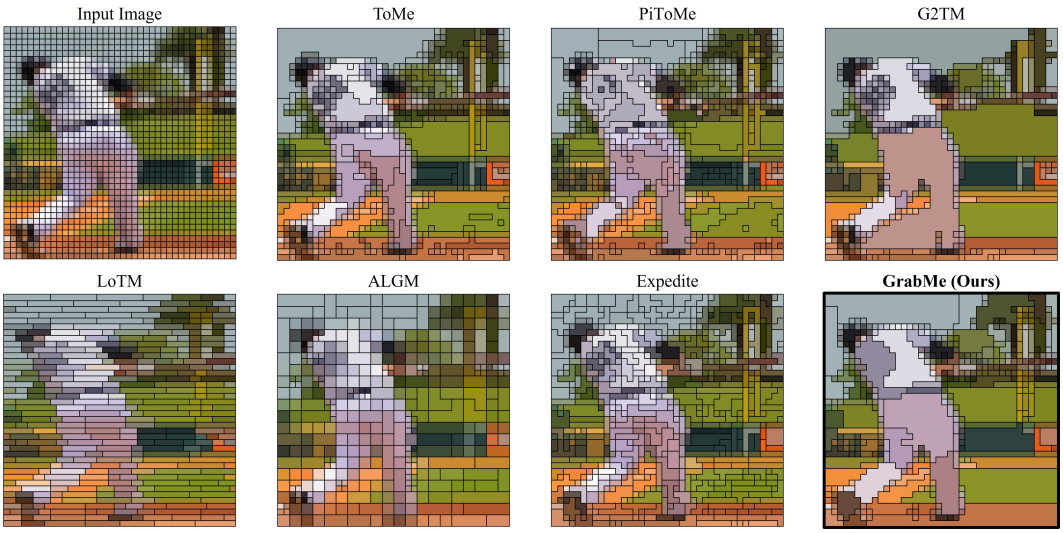



Figure 5: **Visualization of token merging** with Segmenter-Large at a matched token reduction rate. Flexible graph connectivity of GrabMe effectively adapts to complex region shapes. Moreover, GrabMe suppresses merging near region boundaries, thereby preserving fine-grained spatial details.

## 5 Discussion and Limitations

Our results demonstrate that GrabMe effectively improves the accuracy–efficiency trade-off of pretrained ViT backbones through spatially constrained, context-aware token merging. Its efficient parallel merging preserves relative spatial structure and fine-grained details while forming semantically coherent token groups. As a result, GrabMe supports more aggressive token reduction and higher throughput across semantic, instance, and panoptic segmentation without additional training.

**Limitations.** Nonetheless, our study has a residual limitation in instance-aware segmentation. Unlike semantic segmentation, instance and panoptic segmentation must distinguish different instances within the same semantic class. Although GrabMe remains more robust under aggressive token reduction, it does not explicitly account for instance identity in its merge decisions. Cross-instance merges may therefore occur when tokens from different instances are not yet sufficiently separated. While such cases are rare among merged groups, we found that they appear in about 20% of evaluated images, motivating the incorporation of instance-awareness into merging strategies. We provide a detailed analysis and qualitative examples of cross-instance merging in Appendix C.2.

**Future Work.** One direction for future work is to incorporate instance-level cues into merge decisions, preserving semantic coherence while distinguishing individual objects. As our work focuses on training-free inference-time acceleration, another promising direction is to apply GrabMe during the pretraining stage, encouraging the backbone to learn merge-aware representations while potentially reducing training cost. Finally, we leave further investigation of whether the merging behavior of GrabMe generalizes to computer vision tasks beyond segmentation for future work.

## 6 Conclusion

In this paper, we introduced GrabMe, a training-free graph-based token merging method for segmentation. GrabMe restricts token merging to one-hop neighbors on a spatial adjacency graph and progressively updates graph connectivity by inheriting the adjacencies of their constituent tokens. By pairing maximum-cardinality conflict-free candidate selection with context-aware merging, GrabMe enables efficient local token merging with reliable preservation of fine-grained details. Across semantic, instance, and panoptic segmentation benchmarks with several ViT backbones, GrabMe effectively improves the accuracy–efficiency trade-off, achieving more aggressive token reduction and higher throughput while better preserving task performance, all without any additional training.

### AI USE STATEMENT

Generative AI tools were used at several stages of this work, including drafting and revising the manuscript, searching for and summarizing relevant literature, and assisting with experimental and implementation code. AI-generated suggestions were used as supporting material rather than as authoritative outputs. The authors independently determined the methodology and experimental design, reviewed and tested AI-assisted code, verified references against the original publications, and checked all reported results and claims. All AI-assisted text was reviewed and revised before inclusion in the manuscript. The authors take full responsibility for the final content of this work.

### REPRODUCIBILITY STATEMENT

To facilitate reproduction of our results, we provide an anonymized implementation of GrabMe together with the code used for evaluation. The complete merging procedure and its pseudocode are described in Appendix A.1, while the exact reduction settings and baseline configurations are provided in Appendix A.2. The datasets, backbones, evaluation metrics, and experimental protocol are specified in Section 4, with computing resources and measurement conditions documented in Appendix A.3. Additional reduction-rate sweeps and efficiency analyses are provided in Appendix B.

## REFERENCES

Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. SegNet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. doi: 10.1109/TPAMI.2016.2644615.

Victor Bercy, Martyna Poreba, Michal Szczepanski, and Samia Bouchafa. G2TM: Single-module graph-guided token merging for efficient semantic segmentation. In *VISAPP 2026-Computer Vision Theory and Applications*, volume 2, pp. 43–54, 2026.

Benjamin Bergner, Christoph Lippert, and Aravindh Mahendran. Token cropr: Faster vits for quite a few tasks. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pp. 9740–9750, 2025.

Daniel Bolya, Cheng-Yang Fu, Xiaoliang Dai, Peizhao Zhang, Christoph Feichtenhofer, and Judy Hoffman. Token merging: Your ViT but faster. In *The Eleventh International Conference on Learning Representations*, 2023.

Holger Caesar, Jasper Uijlings, and Vittorio Ferrari. COCO-stuff: Thing and stuff classes in context. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1209–1218. IEEE, 2018.

Gen-Huey Chen, MT Kuo, and JP Sheu. An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT Numerical Mathematics*, 28(2):353–356, 1988.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4): 834–848, 2018. doi: 10.1109/TPAMI.2017.2699184.

Mengzhao Chen, Wenqi Shao, Peng Xu, Mingbao Lin, Kaipeng Zhang, Fei Chao, Rongrong Ji, Yu Qiao, and Ping Luo. Diffrate: Differentiable compression rate for efficient vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 17164–17174, 2023a.

Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. In *The Eleventh International Conference on Learning Representations*, 2023b.

Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1290–1299, 2022.

Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

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.

Yuxin Fang, Wen Wang, Binhui Xie, Quan Sun, Ledell Wu, Xinggang Wang, Tiejun Huang, Xinlong Wang, and Yue Cao. EVA: Exploring the limits of masked visual representation learning at scale. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 19358–19369, 2023.

Mohsen Fayyaz, Soroush Abbasi Koohpayegani, Farnoush Rezaei Jafari, Sunando Sengupta, Hamid Reza Vaezi Joze, Eric Sommerlade, Hamed Pirsiavash, and Jürgen Gall. Adaptive token sampling for efficient vision transformers. In *European conference on computer vision*, pp. 396–414. Springer, 2022.

Zhanzhou Feng and Shiliang Zhang. Efficient vision transformer via token merger. *IEEE Transactions on Image Processing*, 32:4156–4169, 2023.

Karim Haroun, Jean Martinet, Karim Ben Chehida, and Thibault Allenet. Leveraging local similarity for token merging in vision transformers. In *International Conference on Neural Information Processing*, pp. 286–300, 2024.

Karim Haroun, Thibault Allenet, Karim Ben Chehida, and Jean Martinet. Dynamic hierarchical token merging for vision transformers. In *VISAPP-2025-20th international joint conference on computer vision, imaging and computer graphics theory and applications*, pp. 677–684, 2025.

Joakim Bruslund Haurum, Sergio Escalera, Graham W Taylor, and Thomas B Moeslund. Agglomerative token clustering. In *European Conference on Computer Vision*, pp. 200–218. Springer, 2024.

Tommie Kerssies, Niccolo Cavagnero, Alexander Hermans, Narges Norouzi, Giuseppe Averta, Bastian Leibe, Gijs Dubbelman, and Daan De Geus. Your ViT is secretly an image segmentation model. In *Proceedings of the computer vision and pattern recognition conference*, pp. 25303–25313, 2025.

Minchul Kim, Shangqian Gao, Yen-Chang Hsu, Yilin Shen, and Hongxia Jin. Token fusion: Bridging the gap between token pruning and token merging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1383–1392, 2024.

Zhenglun Kong, Peiyan Dong, Xiaolong Ma, Xin Meng, Wei Niu, Mengshu Sun, Xuan Shen, Geng Yuan, Bin Ren, Hao Tang, et al. SPViT: Enabling faster vision transformers via latency-aware soft token pruning. In *European conference on computer vision*, pp. 620–640. Springer, 2022.

Sanghyeok Lee, Joonmyung Choi, and Hyunwoo J Kim. Multi-criteria token fusion with one-step-ahead attention for efficient vision transformers. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 15741–15750. IEEE, 2024.

Jin Li, Yaoming Wang, Xiaopeng Zhang, Bowen Shi, Dongsheng Jiang, Chenglin Li, Wenrui Dai, Hongkai Xiong, and Qi Tian. AiluRus: A scalable ViT framework for dense prediction. In *NeurIPS*, 2023.

Weicong Liang, Yuhui Yuan, Henghui Ding, Xiao Luo, Weihong Lin, Ding Jia, Zheng Zhang, Chao Zhang, and Han Hu. Expediting large-scale vision transformer for dense prediction without fine-tuning. In *NeurIPS*, 2022.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

Chenyang Lu, Daan De Geus, and Gijs Dubbelman. Content-aware token sharing for efficient semantic segmentation with vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 23631–23640, 2023.

Narges Norouzi, Svetlana Orlova, Daan De Geus, and Gijs Dubbelman. ALGM: Adaptive local-then-global token merging for efficient semantic segmentation with plain vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 15773–15782, 2024.

Yongming Rao, Wenliang Zhao, Benlin Liu, Jiwen Lu, Jie Zhou, and Cho-Jui Hsieh. DynamicViT: Efficient vision transformers with dynamic token sparsification. In *Advances in Neural Information Processing Systems*, 2021.

Robin Strudel, Ricardo Garcia, Ivan Laptev, and Cordelia Schmid. Segmenter: Transformer for semantic segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 7262–7272, 2021.

Quan Tang, Bowen Zhang, Jiajun Liu, Fagui Liu, and Yifan Liu. Dynamic token pruning in plain vision transformers for semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 777–786, 2023.

Chau Tran, Duy M. H. Nguyen, Manh-Duy Nguyen, TrungTin Nguyen, Ngan Le, Pengtao Xie, Daniel Sonntag, James Y. Zou, Binh Nguyen, and Mathias Niepert. Accelerating transformers with spectrum-preserving token merging. In *NeurIPS*, 2024.

Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 568–578, 2021.

Yancheng Wang and Yingzhen Yang. Efficient visual transformer by learnable token merging. *IEEE transactions on pattern analysis and machine intelligence*, 2025.

Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, José M. Álvarez, and Ping Luo. SegFormer: Simple and efficient design for semantic segmentation with transformers. In *NeurIPS*, pp. 12077–12090, 2021.

Hongxu Yin, Arash Vahdat, Jose M Alvarez, Arun Mallya, Jan Kautz, and Pavlo Molchanov. A-ViT: Adaptive tokens for efficient vision transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10809–10818, 2022.

Bowen Zhang, Zhi Tian, Quan Tang, Xiangxiang Chu, Xiaolin Wei, Chunhua Shen, and Yifan Liu. SegViT: Semantic segmentation with plain vision transformers. In *NeurIPS*, 2022.

Xin Zhang, Weiying Xie, Yunsong Li, Xiaoyu Chen, Tianlin Hui, Jitao Ma, and Leyuan Fang. TF-ATM: Training-free adaptive token merging. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 7172–7180, 2025.

Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890, 2017.

Sixiao Zheng, Jiachen Lu, Hengshuang Zhao, Xiatian Zhu, Zekun Luo, Yabiao Wang, Yanwei Fu, Jianfeng Feng, Tao Xiang, Philip HS Torr, et al. Rethinking semantic segmentation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 6881–6890, 2021.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 633–641, 2017.

# APPENDIX TABLE OF CONTENTS

**A Experimental Details** **14**

A.1 Pseudocode of GrabMe . . . . . 14

A.2 Configurations and Reduction Rates . . . . . 15

A.3 Computing Resources and Measurement Protocol . . . . . 15

**B Additional Experiments** **16**

B.1 Reduction-Rate Sweep on Semantic Segmentation . . . . . 16

B.2 Comparison on a Smaller Backbone . . . . . 17

B.3 FLOPs and Memory . . . . . 17

B.4 Further Analysis of DP-based Selection . . . . . 18

B.5 Unmerge Efficiency . . . . . 18

B.6 Batch-Size Scaling . . . . . 19

**C Additional Visualization** **19**

C.1 Token-Group Visualization . . . . . 19

C.2 Instance-Aware Failure Cases . . . . . 19

C.3 Visualization of Progressive Token Merging of GrabMe . . . . . 21

C.4 Additional Context-Score and Token-Merging Visualizations . . . . . 21

# A Experimental Details

We provide additional implementation details, experimental configurations, and computational-resource information for the experiments reported in the main paper. We use $k$ for the per-layer merge budget, consistent with the main text, and $\rho$ for the token reduction rate.

## A.1 Pseudocode of GrabMe

GrabMe maintains token representations $\mathbf{X}$, group sizes $\mathbf{m}$, and a group identifier $\mathbf{g}$ that maps each original patch to its current token group. Algorithm 1 summarizes a single merge step. Each token first computes feature similarities with its one-hop neighbors and identifies the most similar one as its merge target. GrabMe then constructs a maximum-cardinality conflict-free candidate pool using dynamic programming. Candidate pairs are ranked by pair similarity and context scores, with the top-$k$ pairs merged in parallel. Multiple source tokens may share the same merge target.

**Algorithm 1** One merge step of GrabMe

**Require:** Token representations $\mathbf{X}$, group sizes $\mathbf{m}$, group identifiers $\mathbf{g}$, original grid edges $\mathcal{E}_0$, merge budget $k$

Normalize each token representation: $\mathbf{z}_u \leftarrow \mathbf{x}_u / \|\mathbf{x}_u\|_2$
Construct the current adjacency multiset $\mathcal{H} \leftarrow \{(g_p, g_q) : (p, q) \in \mathcal{E}_0,\ g_p \neq g_q\}$
**for** each current token $u$ **do**
    $\mathcal{N}(u) \leftarrow$ one-hop neighbors of $u$ in $\mathcal{H}$
    $s_{uv} \leftarrow \mathbf{z}_u^\top \mathbf{z}_v$ for $v \in \mathcal{N}(u)$
    $d(u) \leftarrow \arg\max_{v \in \mathcal{N}(u)} s_{uv}$
    $q_{uv} \leftarrow (s_{uv} + 1)/2$
    $\mu_u, \sigma_u \leftarrow \mathrm{mean}_v\, q_{uv}, \mathrm{std}_v\, q_{uv}$
    $c_u \leftarrow \mu_u - \sigma_u$
**end for**
$\mathcal{S}^* \leftarrow$ MaximumCandidatePool$(d)$ ▷ Algorithm 2
**for** each $u \in \mathcal{S}^*$ **do**
    $v \leftarrow d(u)$
    $R_{u,v} \leftarrow \frac{1}{2}\left(q_{uv} + \frac{c_u + c_v}{2}\right)$
**end for**
$\mathcal{P} \leftarrow$ top-$k$ pairs $\{(u, d(u)) : u \in \mathcal{S}^*\}$ ranked by $R_{u,d(u)}$
**for** each surviving merge target $v$ **do**
    $\mathcal{A}_v \leftarrow \{v\} \cup \{u : (u, v) \in \mathcal{P}\}$
    $m'_v \leftarrow \sum_{u \in \mathcal{A}_v} m_u$
    $\mathbf{x}'_v \leftarrow \frac{\sum_{u \in \mathcal{A}_v} m_u \mathbf{x}_u}{m'_v}$
**end for**
Compact the surviving tokens and update $\mathbf{g}$
**return** $\mathbf{X}'$, $\mathbf{m}'$, and $\mathbf{g}$

The spatial graph is obtained by mapping the original four-connected patch-grid edges through the group identifiers, removing intra-group edges while retaining inter-group edges. When multiple original edges connect the same pair of current groups, their multiplicity is retained when computing the local similarity distribution. After merging, the group identifiers are updated accordingly and implicitly determine the graph used at the next merge layer. The same identifiers are used for the *unmerge* operation described in Section 3.1.

Algorithm 2 details the DP-based construction of the maximum-cardinality conflict-free candidate pool. Since every token selects at most one merge target, the resulting directed graph decomposes into cycles with attached trees. The DP first solves the attached trees and then the cycles.

When $u$ is selected, all $v$ with $d(v) = u$ are excluded; otherwise, each independently takes its better state. For cycles, both states of an arbitrary reference token are evaluated to enforce the closing-edge constraint. Ties are broken by accumulated proposal similarity, and the overall complexity is $O(N)$ since each token is processed a constant number of times.

**Algorithm 2** Maximum-cardinality conflict-free candidate pool

**Require:** Merge-target map $d$

Construct edges $u \to d(u)$ and iteratively remove zero-indegree nodes to identify cycles and obtain a leaf-to-root order
**for** each non-cycle node $u$ in this order **do**
    $D_u^1 \leftarrow 1 + \sum_{v:d(v)=u} D_v^0$
    $D_u^0 \leftarrow \sum_{v:d(v)=u} \max\{D_v^0, D_v^1\}$
**end for**
**for** each cycle node $p$ **do**
    $C_p^1 \leftarrow 1 + \sum_{\substack{v:d(v)=p \\ v \notin \text{cycle}}} D_v^0$
    $C_p^0 \leftarrow \sum_{\substack{v:d(v)=p \\ v \notin \text{cycle}}} \max\{D_v^0, D_v^1\}$
**end for**
**for** each directed cycle **do**
    Evaluate an arbitrary reference token as selected and excluded; in the former case, exclude both adjacent cycle tokens
    Retain the higher-cardinality valid state
**end for**
Backtrack the optimal states and return their union $\mathcal{S}^*$

### A.2 Configurations and Reduction Rates

For fixed-budget GrabMe runs with $N_0$ initial patch tokens and $L$ merge layers, the reduction rate is

$$\rho = 100\,\frac{Lk}{N_0}, \tag{7}$$

provided that the requested budget remains feasible at every merge layer. Table 5 summarizes the GrabMe configurations and selected baseline controls used in the experiments.

Table 5: **Baseline configurations.** Methods are grouped by their token-reduction control: a similarity threshold $\tau$, the number of superpixels, or a budget $k$ for constant-based methods and GrabMe.

| Method | Segmenter-Large | | | EoMT | | EVA & ViT-Adapter |
|---|---|---|---|---|---|---|
| | ADE20K | Cityscapes | COCO-Stuff | ADE20K | COCO | ADE20K |
| *Threshold $\tau$* | | | | | | |
| + ALGM | 0.96 | 1.0 | 0.97 | 0.98 | 0.65 | 0.98 |
| + G2TM | 0.97 | 0.985 | 0.99 | 0.995 | 0.82 | 0.98 |
| *superpixels* | | | | | | |
| + Expedite | 841 | 2116 | 625 | 625 | 324 | 1849 |
| *Constant $k$* | | | | | | |
| + ToMe | 26 | 38 | 3 | 1 | 64 | 31 |
| + PiToMe | 7 | 10 | 47 | 2 | 64 | 8 |
| + MCTF | 28 | 40 | 49 | – | – | – |
| + ATC | 29 | 29 | 18 | – | – | – |
| + DHTM | 107 | 77 | 245 | – | – | – |
| + LoTM | 160 | 100 | 5 | 51 | 320 | 784 |
| + GrabMe (Ours) | 46 | 43 | 48 | 19 | 64 | 59 |

### A.3 Computing Resources and Measurement Protocol

All experiments were conducted using a single NVIDIA RTX A6000 GPU with 48 GB of VRAM. All reported measurements are averaged over three independent runs. Throughput measurements use 10 warm-up iterations followed by 30 timed forward passes, with batch size 32 unless otherwise specified; for EVA, batch size 8 is used due to GPU memory constraints. Data loading, preprocessing, and host-to-device transfer are excluded from timing.

For semantic segmentation, accuracy is measured by mean intersection-over-union (mIoU), obtained by averaging the class-wise IoU between predicted and ground-truth masks. For instance segmenta-

tion, we report COCO mask AP ($AP_{mask}$), averaged over IoU thresholds from 0.50 to 0.95, together with $AP_{75}$ at an IoU threshold of 0.75 and $AP_S$ for small objects, defined in COCO as those with an area below $32^2$ pixels. Panoptic segmentation is evaluated using panoptic quality (PQ), which accounts for both recognition and segmentation quality; $PQ^{Th}$ and $PQ^{St}$ denote PQ over thing and stuff classes, respectively. Higher values indicate better performance for all segmentation metrics. For Tables 1 and 2, backbone mIoU is considered preserved when the mean mIoU over three runs, rounded to one decimal place, matches that of the original backbone. The same criterion is applied to GrabMe and all baselines.

# B Additional Experiments

## B.1 Reduction-Rate Sweep on Semantic Segmentation

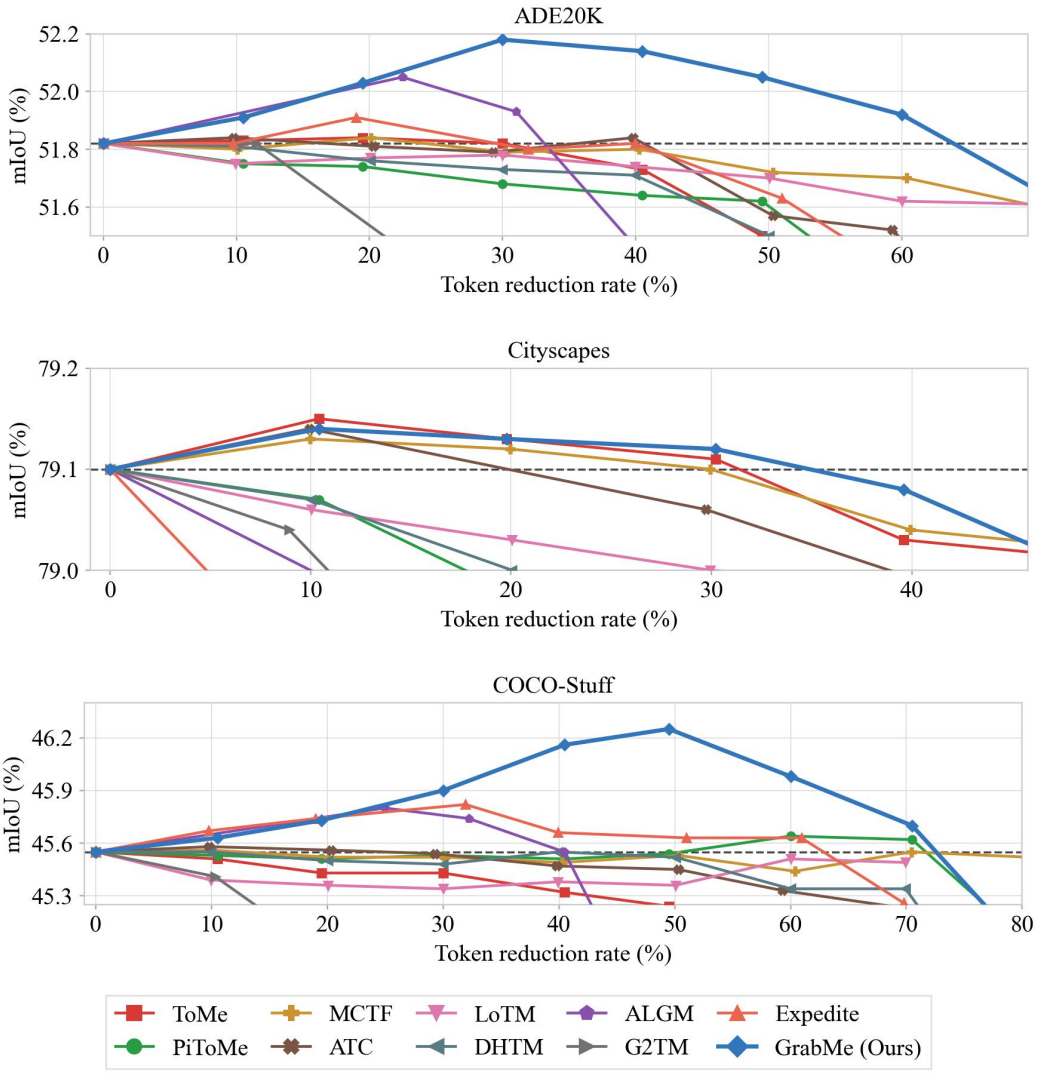



Figure 6: **Accuracy–efficiency trade-off on semantic segmentation.** mIoU versus token-reduction rate using Segmenter-Large on ADE20K, Cityscapes, and COCO-Stuff. The plotted points are evaluated at approximately 10-percentage-point intervals, and the connecting lines are shown only to visualize the overall trend rather than continuous performance curves.

For the semantic segmentation comparison in Table 1, we report for each baseline the maximum token-reduction rate that preserves the backbone mIoU; when this criterion cannot be met, we in-

Table 6: **Comparison on ADE20K with Segmenter-Small.** Each method is evaluated at the highest token-reduction rate that preserves the backbone mIoU; otherwise, the operating point with the highest mIoU is reported and highlighted in red. Throughput is measured with batch size 32.

| Method | mIoU | rate (%) | im/s (↑) |
|---|---|---|---|
| Segmenter-Small | 45.3 | 0.0 | 113 |
| + ToMe | 45.3 | 11.7 | 103 |
| + PiToMe | 45.3 | 7.0 | 105 |
| + LoTM | 45.3 | 7.9 | 117 |
| + ALGM | 45.3 | 35.6 | 121 |
| + G2TM | 45.3 | 0.4 | 90 |
| + Expedite | 41.1 | 6.2 | 90 |
| + GrabMe (Ours) | 45.3 | **39.8** | **149** |

stead report the reduction rate achieving the highest mIoU. To determine the operating points, we first evaluate GrabMe and all baselines at approximately 10-percentage-point reduction intervals, and then refine the search at 1-percentage-point intervals around the highest reduction rate that reaches the backbone mIoU. The resulting accuracy–reduction curves are summarized in Figure 6.

### B.2 Comparison on a Smaller Backbone

We further evaluate GrabMe with Segmenter-Small on ADE20K to examine whether its effectiveness extends to a smaller backbone. For each method, we report the highest token-reduction rate that preserves the backbone mIoU; when this criterion cannot be met, we report the highest achieved mIoU and its corresponding reduction rate. Throughput is measured with batch size 32.

GrabMe preserves the backbone mIoU at 39.8% token reduction while increasing throughput from 113 to 149 im/s, corresponding to approximately 30% higher throughput. It also supports a higher reduction rate than the competing methods that retain the backbone accuracy. This setting is particularly challenging because the backbone itself is computationally lighter, making the overhead of token selection and merging a larger fraction of the overall inference cost. Nevertheless, GrabMe achieves a substantial speedup through aggressive token reduction and efficient parallel merging. These results show that the accuracy–efficiency gains observed with the large backbones in Table 1 are not limited to high-capacity models, but extend to a substantially smaller backbone.

### B.3 FLOPs and Memory

Table 7: **GFLOPs and GPU memory usage using Segmenter-Large on ADE20K.** Methods are evaluated at the maximum token reduction rate that preserves backbone performance.

| Method | GFLOPs | Peak Allocated GiB | Mean Allocated GiB |
|---|---|---|---|
| Segmenter-Large | 671.7 | 14.3 | 8.7 |
| *Global-range merging* | | | |
| + ToMe | 511.2 | **12.4** | 1.9 |
| + PiToMe | 637.3 | 12.8 | 2.4 |
| + MCTF | 674.6 | 17.3 | 4.7 |
| + ATC | 472.0 | **12.4** | 2.0 |
| *Local-range merging* | | | |
| + LoTM | 584.8 | **12.4** | 2.4 |
| + DHTM | 613.0 | **12.4** | 2.1 |
| + G2TM | 666.6 | 14.3 | 7.5 |
| + ALGM | 533.8 | **12.4** | 7.1 |
| + Expedite | 470.8 | 12.5 | 8.9 |
| + GrabMe (Ours) | **397.0** | **12.4** | **1.8** |

We evaluate computational cost and GPU memory using Segmenter-Large on ADE20K at each method's selected operating point. GrabMe requires 397.0 GFLOPs per image, reducing the counted computation by approximately 40% relative to the original backbone at 671.7 GFLOPs.

GFLOPs are measured using `fvcore`, with one MAC counted as one operation. Unsupported operations, including custom merging kernels, are excluded from the FLOP count; GFLOPs should therefore be regarded as a complementary measure of computational cost rather than a complete accounting of runtime operations. Peak memory denotes the maximum CUDA-allocated memory, while mean memory is sampled at approximately 1 ms intervals over three forward passes.

### B.4 Further Analysis of DP-based Selection

Table 8: **Comparison of candidate selection strategies** on ADE20K with Segmenter-Large. Pool size denotes the layer-wise average number of conflict-free candidate pairs eligible for parallel merging. Throughput (im/s) is measured with batch size 32.

| Method | Pool Size | Feature Similarity | Semantic Consistency | im/s |
|---|---|---|---|---|
| Greedy Selection | 535.0 | **0.964** | 0.935 | 13.17 |
| Bipartite Partition | 534.8 | 0.919 | 0.966 | **15.63** |
| DP Selection (Ours) | **614.3** | 0.920 | **0.986** | 15.56 |

We further analyze the candidate pools underlying the candidate-selection ablation in Table 4b. Table 8 compares the average pool size, feature similarity, and semantic consistency on ADE20K.

DP-based selection constructs the largest candidate pool among the methods, increasing the pool size from 535.0 and 534.8 for greedy and bipartite selection, respectively, to 614.3. Compared with bipartite partitioning, it maintains similar feature similarity while achieving higher semantic consistency. Greedy selection yields the highest feature similarity, but produces a smaller pool with lowest semantic consistency. At a fixed 69.0% reduction rate, greedy, bipartite, and DP selection achieve 51.3, 51.5, and 51.8 mIoU, respectively (Tab. 4b), with throughputs of 13.17, 15.63, and 15.56 im/s. These results show that DP-based candidate selection improves candidate quality with a larger feasible pool while introducing negligible overhead compared with simple bipartite partitioning.

### B.5 Unmerge Efficiency

Table 9: **Unmerge efficiency.** Measurements use EVA with ViT-Adapter at batch size 8. Columns report the unmerge latency after the indicated number of merge layers, including restoration-state updates; memory denotes the restoration state stored for unmerging after 40 merge layers.

| Method | Unmerge Latency (ms, ↓) | | | | | Memory (MiB, ↓) |
|---|---|---|---|---|---|---|
| | Layer 1 | Layer 5 | Layer 10 | Layer 20 | Layer 40 | |
| ToMe | **0.32** | 1.44 | 2.73 | 4.88 | 7.51 | 4.85 |
| ALGM | 3.41 | 10.10 | 17.67 | 30.22 | 44.93 | 62.78 |
| Expedite | 26.76 | 123.6 | 223.6 | 364.3 | 471.8 | 2429 |
| GrabMe (Ours) | 0.35 | **0.47** | **0.59** | **0.83** | **1.28** | **0.19** |

Decoders often require a dense sequence at the original patch resolution to predict segmentation masks. GrabMe therefore maintains a group identifier for each original patch, indicating its current token group, and restores the dense sequence through a single $\mathrm{gather}$ operation, assigning each original patch $p$ the feature of its corresponding group, $\mathbf{x}_p^{\mathrm{dense}} = \mathbf{x}_{g_p}^{\mathrm{compact}}$. Table 9 compares the unmerge efficiency of GrabMe with other methods in terms of latency across merge layers and restoration-state memory after 40 layers. In the main experiments, baselines without a dedicated unmerge mechanism adopt restoration scheme of ToMe.

ToMe recursively reverses stored merge mappings, causing its restoration cost to grow with the number of merge layers. ALGM maintains a dense source membership matrix with storage complexity $O(B \times N \times G)$, where $B$, $N$, and $G$ denote the batch size, number of original patch tokens, and number of current token groups, respectively. Expedite employs a separate reconstruction layer that recomputes token-to-supertoken affinities, resulting in additional computation and storage. In contrast, GrabMe stores only one group index for each original patch, requiring $O(B \times N)$ memory,

and reconstructs the dense sequence with a single lightweight gather operation. Consequently, after 40 merge layers, unmerging requires only 1.28 ms and 0.19 MiB of restoration-state memory.

Efficient unmerging is particularly important for architectures such as ViT-Adapter (Chen et al., 2023b), whose intermediate interaction stages require features at the original sequence resolution. In such cases, restoration overhead can offset the gains from token reduction. GrabMe's lightweight restoration helps preserve these efficiency gains across modern ViT-based segmentation backbones.

### B.6 Batch-Size Scaling

We evaluate GrabMe with Segmenter-Large on ADE20K across batch sizes that fit within the available memory constraints, while keeping the token-reduction rate fixed at 69.0%.

As shown in Table 10, throughput increases with batch size, reflecting the efficiency of parallel merging. This behavior contrasts with threshold-based merging methods, where a fixed threshold can produce different numbers of merged tokens across samples. Batched execution therefore requires either padding or matching all samples to the least-reduced sample, which may limit the effective token reduction as batch size grows. GrabMe instead removes the same $k$ tokens from every sample, maintaining a uniform sequence length without such constraints. This fixed-budget design allows its parallel merging to benefit effectively from larger batch sizes.

Table 10: **Batch-size scaling** for GrabMe at 69.0% token reduction.

| Batch size | im/s |
|---|---|
| 1 | 12.7 |
| 2 | 13.7 |
| 4 | 14.6 |
| 8 | 15.2 |
| 16 | 15.3 |
| **32** | **15.6** |

## C Additional Visualization

### C.1 Token-Group Visualization

Figure 7 illustrates how the group size $m$ and group identifier $g$ described in Section 3.1 are updated during progressive merging. When a source token $u$ is merged into its target $d(u)$, the target group size is updated as $m_{d(u)} \leftarrow m_u + m_{d(u)}$. At the same time, the group identifiers of all original patches previously assigned to $u$ are updated to point to $d(u)$. Repeating this process allows GrabMe to maintain both the size of each current token group and the correspondence from every original patch to its current group throughout the encoder. For clearer illustration, the figure uses a sample image divided into a $10\times10$ grid and applies GrabMe using the average RGB value of each patch rather than the learned token representations produced by the pretrained backbone.

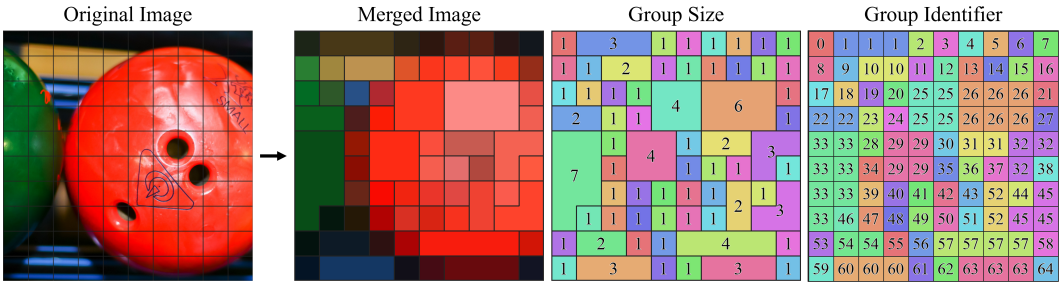



Figure 7: **Illustration of group size and group identifiers.** From left to right, we show the input image, the merged image where each group is shown by its average RGB value, the resulting group sizes, and the group identifiers of the original patches.

### C.2 Instance-Aware Failure Cases

GrabMe does not explicitly incorporate instance identity into its merge decisions. Consequently, adjacent objects belonging to the same semantic class but different instances may occasionally be merged. To quantify this behavior, we first assign each original patch to a ground-truth instance

Table 11: **Cross-instance merging on COCO** using EoMT-Large at an 80% token-reduction rate.

| Task | Valid images | Affected images | Conflicting groups (%) |
|---|---|---|---|
| Instance segmentation | 4952 | 876 | 0.43 |
| Panoptic segmentation | 5000 | 1035 | 0.50 |

when a single non-crowd thing instance occupies more than half of the patch area. We define a conflicting group as a merged token group that contains patches from two or more different ground-truth instances of the same semantic class. An image is considered affected if it contains at least one such group.

As shown in Tab 11, conflicting groups account for only 0.43% and 0.50% of the resulting token groups for instance and panoptic segmentation, respectively, indicating that such cross-instance merges remain relatively rare. However, they affect 17.69% of instance-segmentation images and 20.70% of panoptic-segmentation images. From this perspective, incorporating instance-level signals into merge decisions may be a promising direction for further improving token merging in instance-aware segmentation tasks. Figure 8 provides qualitative examples of adjacent instances from the same semantic class being merged into a single token group.

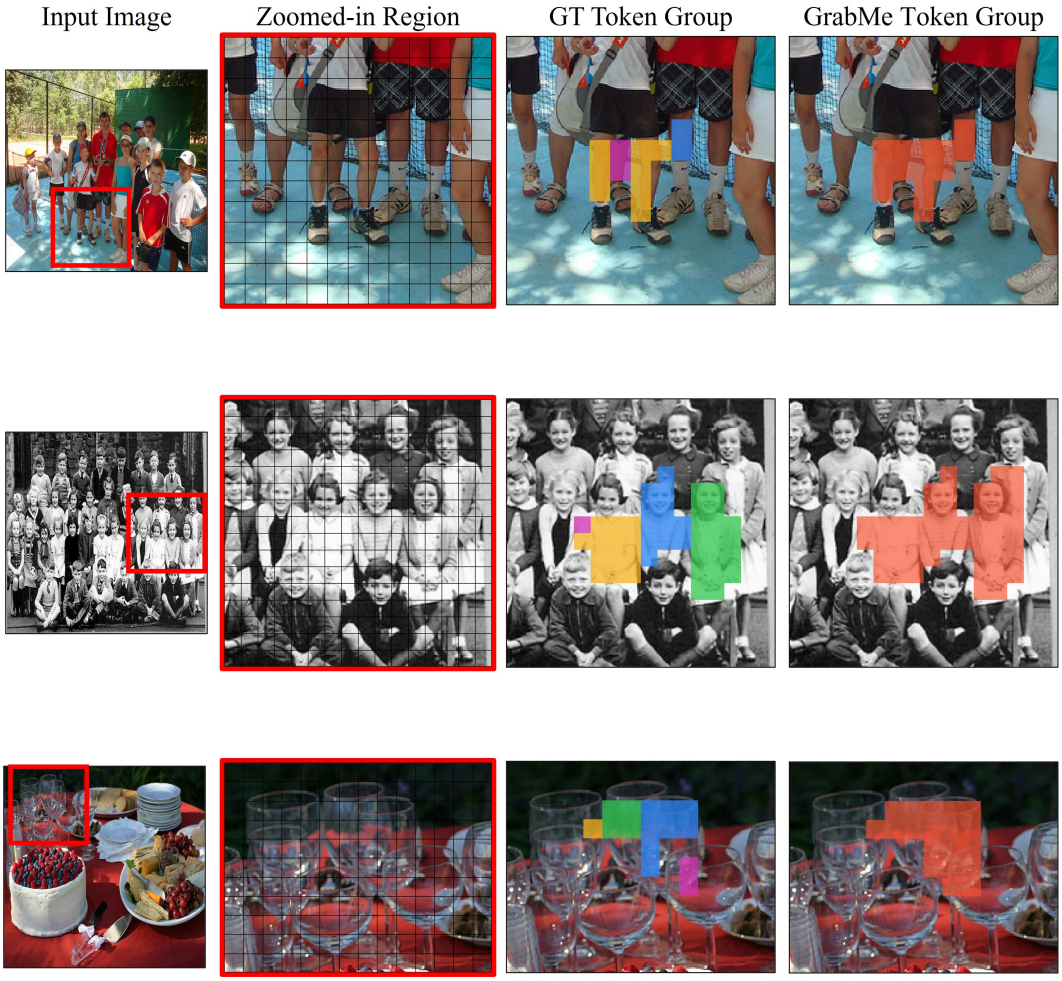



Figure 8: **Qualitative examples of cross-instance merging.** We compare ground-truth instance groupings with GrabMe token groups in regions containing adjacent instances of the same semantic class. Each color denotes a distinct token group.

### C.3 Visualization of Progressive Token Merging of GrabMe

GrabMe progressively merges tokens across encoder layers while dynamically updating the spatial graph. After each merge, the resulting group inherits the one-hop adjacencies of its constituent tokens, removing intra-group edges while retaining inter-group connections. Consequently, graph connectivity gradually expands as merging proceeds, allowing token groups to flexibly adapt to regions with complex and irregular shapes while remaining spatially contiguous. This dynamic graph-based local merging is itself effective for segmentation: As shown by the context-free variant in Table 1, GrabMe remains competitive with existing token-merging baselines even without context-aware ranking. Incorporating the context score further improves its effectiveness by discouraging merges in spatially sensitive regions. Figure 9 visualizes this progressive merging process across layers, illustrating how token groups gradually grow into larger, spatially coherent regions.

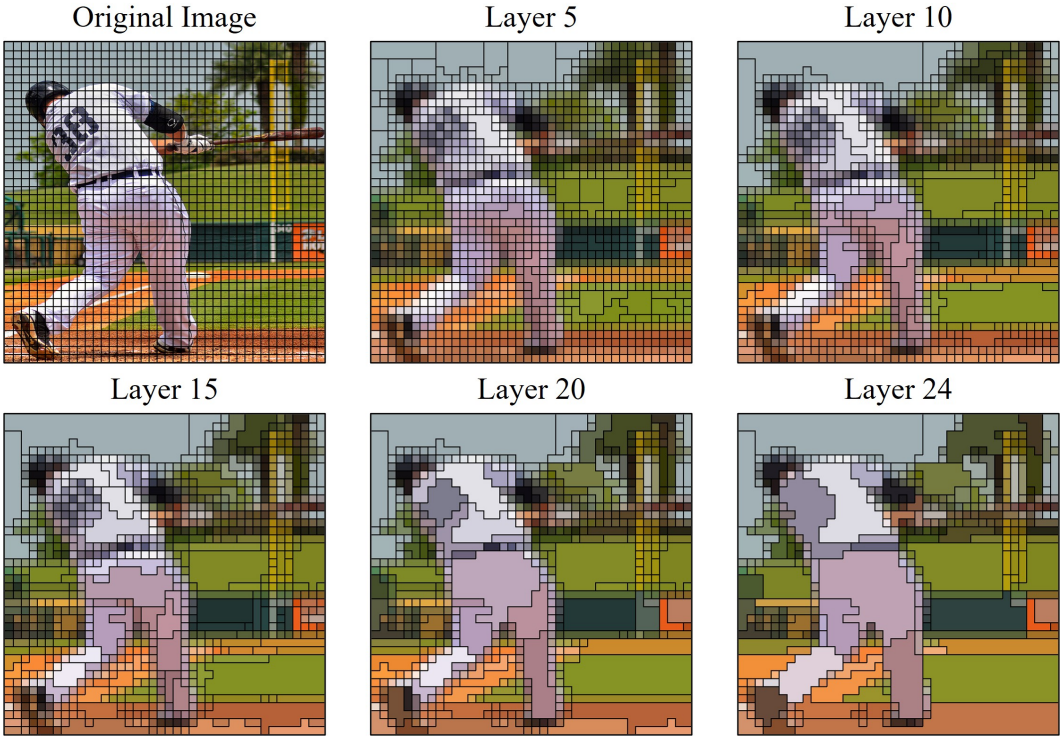



Figure 9: **Layer-wise visualization of progressive token merging with GrabMe.** On Segmenter-Large, GrabMe progressively expands token groups through graph connectivity while maintaining spatial contiguity, thereby retaining the relative spatial structure of the original patches. Its context-aware ranking further suppresses merging near object boundaries and within small, thin structures, preserving fine-grained details important for segmentation.

### C.4 Additional Context-Score and Token-Merging Visualizations

We provide additional visualizations of context-score maps and the corresponding token-merging results of GrabMe on COCO-Stuff with Segmenter-Large. GrabMe is applied at the maximum token-reduction rate that preserves the backbone mIoU, as reported in Table 1. The visualizations are provided in Figure 10 and show that spatially sensitive regions, such as object boundaries and small, thin structures, have lower context scores. Consequently, GrabMe performs token merging while effectively retaining fine-grained details crucial for segmentation.

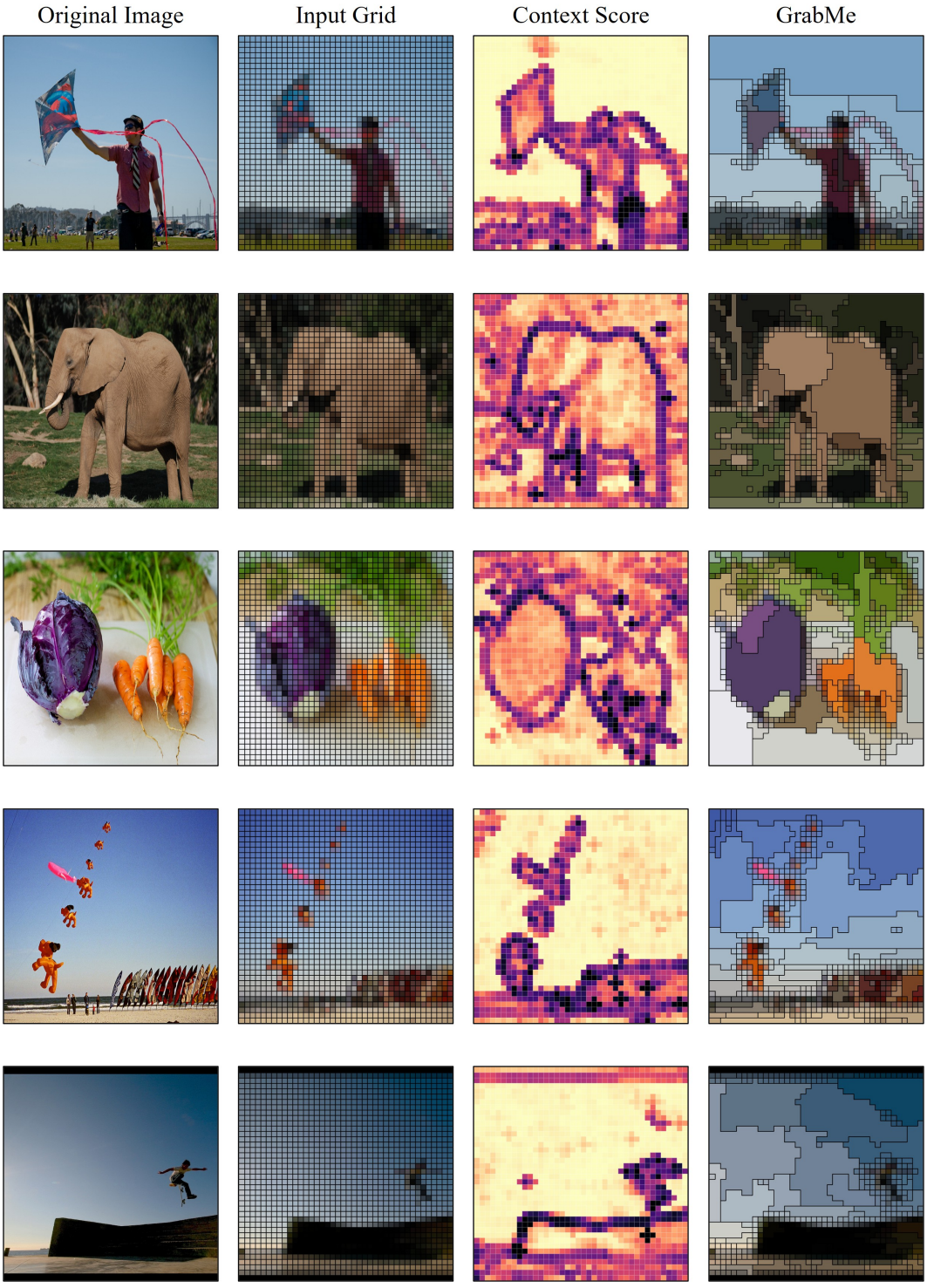



Figure 10: **Context-score and token-merging visualization on COCO-Stuff using Segmenter-Large.** Context-score maps are shown alongside the corresponding GrabMe token groups, with brighter regions indicating higher context scores and darker regions lower scores.