DENSE CHECKPOINTS, SPARSE EXECUTION: RETRAINING-FREE ACCELERATION OF QUERY-BASED SEGMENTATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Query-based segmentation repeatedly materializes dense query-pixel tensors during decoding and output processing, generating substantial memory traffic at high image resolutions. We use a frozen checkpoint's predicted masks and logits as an input-dependent execution plan to reduce this traffic. Mask-guided sparse cross-attention with a fixed budget of 128 feature cells per query preserves dense-level panoptic, instance, and semantic accuracy across compatible checkpoints on Cityscapes, COCO, and ADE20K, without retraining or checkpoint-specific tuning. Coarse intermediate mask prediction at the next attention stride avoids temporary stride-4 masks. Task-sufficient output processing evaluates masks only in regions that can affect the final output. For panoptic segmentation, we find that masks can influence pixel assignments even where they predict background, and derive conditions for safely skipping computation. Given identical FP32 logits from a released Mask2Former R50 checkpoint, output processing exactly reproduces the original panoptic maps, instance masks, and labels on all 500 Cityscapes validation images. On Mask2Former at native Cityscapes resolution, these transformations reduce FP32 DRAM traffic by 67% in the transformer decoder and over 80% in output processing. On an AGX Orin, output processing achieves a $1.8\times$ panoptic speedup and a $4.2\times$ instance speedup over optimized dense processing. With a PPM-FPN pixel decoder under mixed precision, complete-model inference achieves speedups of up to $1.31\times$ for panoptic and $1.54\times$ for instance segmentation over an optimized dense reference. Dense segmentation checkpoints can therefore be deployed more efficiently without retraining by targeting the inference paths that dominate memory traffic.

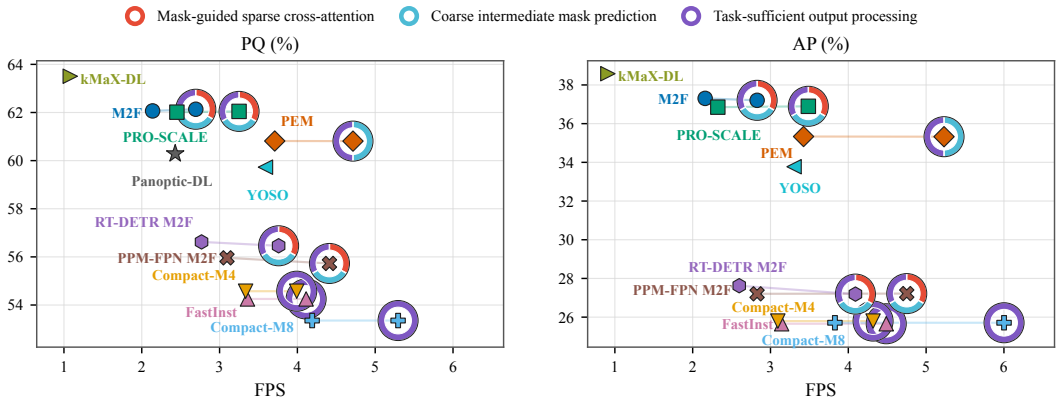


Figure 1: Complete-model accuracy and inference speed on the Cityscapes validation set at native 1024×2048 resolution, using FP32 and batch size one on an AGX Orin. Each marker and color identifies one model. Colored rings mark our transformed execution, with ring colors identifying the enabled components. We use released checkpoints where available and otherwise follow published recipes. All models use R50 except Panoptic-DeepLab, which uses R52-DC5.

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.

1 INTRODUCTION

Query-based segmentation models predict a compact set of object masks, yet repeatedly materialize dense query-pixel tensors during inference (Cheng et al., 2021; 2022). At high image resolutions, dense cross-attention evaluates tens of thousands of feature cells per query. Intermediate masks are predicted at stride 4 before being immediately downsampled for the next attention block, while output processing evaluates query masks across the full image, including regions where their values cannot affect the final instance or panoptic output. Prior work reduces decoder activation traffic on edge GPUs but leaves postprocessing unoptimized, despite finding that it can become the largest remaining component (Busemann et al., 2026).

Many methods improve query-based segmentation efficiency through architectural changes and dedicated training (He et al., 2023b; Cavagnero et al., 2024; Aich et al., 2025; Busemann et al., 2026). SparseOcc demonstrates mask-guided sparse sampling in a trained 3D occupancy architecture (Liu et al., 2024a). We investigate how to reduce computation and memory traffic in dense-trained segmentation models without retraining. Their predicted masks and logits already provide an input-dependent execution plan for making both decoding and output processing more efficient.

We realize this idea through three composable inference transformations. Mask-guided sparse cross-attention replaces dense attention over up to 32,768 feature cells at native Cityscapes resolution with attention to 128 deterministically selected cells per query. Intermediate masks are predicted directly at the next attention block's stride, avoiding stride-4 predictions followed by immediate downsampling. Finally, task-sufficient output processing uses instance-mask support and conservative panoptic score bounds to avoid an intermediate full-resolution floating-point mask stack. We find that queries can still influence panoptic pixel assignments even where their masks predict background, and account for this when pruning. Sparse attention is approximate, coarse prediction is equivalent in real arithmetic, and output processing matches the original instance masks, labels, and panoptic maps in validation with identical class and final mask logits.

We evaluate the applicable transformations across frozen checkpoints from five model families and across panoptic, instance, and semantic segmentation on Cityscapes, COCO, and ADE20K. At native Cityscapes resolution in FP32, task-sufficient output processing achieves a $1.8\times$ panoptic speedup and a $4.2\times$ instance speedup over optimized dense processing on an AGX Orin. Hardware counters on an RTX A4000 show over 80% lower output-processing DRAM traffic than the optimized dense reference. Coarse mask prediction and task-sufficient output processing also reduce inference latency in architectures with already efficient attention, while preserving accuracy (Figure 1).

Our contributions are summarized as follows:

- We show that mask-guided sparse attention with a fixed budget of 128 feature cells per query transfers across compatible frozen segmentation checkpoints, using unchanged settings across datasets and tasks while preserving dense-level accuracy without retraining.
- We develop task-sufficient output processing for instance and panoptic segmentation. For instances, we use predicted mask support to restrict mask evaluation and scoring to regions that can contribute to the output. For panoptic segmentation, we find that masks can affect pixel assignments even where they predict background, and derive conditions for safely skipping computation.
- We combine these methods with coarse intermediate mask prediction for efficient edge deployment, achieving complete-model speedups of up to $1.35\times$ for panoptic and $1.59\times$ for instance segmentation over optimized dense references on an AGX Orin, with similar gains under mixed precision.

2 RELATED WORK

Query-based segmentation. Mask2Former uses predicted masks to restrict which feature cells contribute to each query, but its original implementation still computes attention scores over the full feature grid (Cheng et al., 2022). Efficiency-oriented methods redesign different parts of the pipeline. PRO-SCALE reduces pixel-decoder cost (Aich et al., 2025). FastInst combines an effi-

cient pixel decoder with a dual-path query decoder (He et al., 2023b). PEM replaces dense cross-attention with learned per-query prototype selection (Cavagnero et al., 2024). These methods obtain efficiency through architectural changes and dedicated training. CompactInst identifies decoder activation traffic as a bottleneck for edge inference and reduces this traffic through architectural changes (Busemann et al., 2026). It also identifies mask post-processing as a substantial cost, but excludes it from the reported model timings. Our approach instead transforms frozen checkpoints without retraining and addresses both decoding and post-processing.

Sparse attention and token selection. Deformable DETR attends to a small set of spatial locations using learned sampling offsets and attention weights (Zhu et al., 2021). FASeg uses top-k selection on upsampled attention scores to restrict cross-attention in additional high-resolution decoder blocks (He et al., 2023a). DToP uses intermediate segmentation predictions to let confident encoder tokens exit early; it supports direct application without retraining, although its main results fine-tune the segmentation heads (Tang et al., 2023). ToMe merges similar ViT tokens without retraining (Bolya et al., 2023). Expediting reduces backbone computation in frozen Mask2Former models through token clustering and reconstruction (Liang et al., 2022). SpargeAttention predicts block-level sparsity from queries and keys and combines online softmax filtering with quantization to accelerate language, image, and video generation without retraining (Zhang et al., 2025). SparseOcc samples points within predicted masks and aggregates image features through adaptive mixing in a trained 3D occupancy architecture, including single-frame experiments (Liu et al., 2024a). We select a fixed budget of feature cells for cross-attention in frozen dense-trained image segmentation decoders, preserving their learned attention projections and weights. We additionally quantify the reduction in DRAM traffic and evaluate complete-model latency on an edge GPU.

Efficient mask prediction and output processing. TokenMask predicts ViT masks on the patch grid and upsamples logits rather than features (Galagain et al., 2026). EoMT uses mask annealing during training to eliminate intermediate mask prediction at inference (Kerssies et al., 2025). Our coarse rewrite resizes shared mask features once per target stride and reuses them across decoder layers, avoiding temporary stride-4 masks while leaving final mask prediction unchanged. PointRend and SparseRefine use learned modules to refine selected locations for high-resolution segmentation (Kirillov et al., 2020; Liu et al., 2024b). Our output processing uses existing logits and conservative bounds to avoid unnecessary query-region evaluations under unchanged output rules.

3 METHOD

We accelerate inference in frozen query-based segmentation checkpoints without retraining or per-checkpoint hyperparameter tuning. Model weights remain unchanged. Let Q denote the number of queries and $N_s = H_s W_s$ the number of feature cells at stride s. On Mask2Former, mask-guided sparse cross-attention approximates dense masked attention using a fixed budget of M feature cells per query. Intermediate masks are predicted directly at the next attention block's stride. Output processing retains the original decision rules, using positive support for instance masks and conservative score bounds for panoptic competition (Figure 2). Feature extraction, query self-attention, feed-forward layers, dense stride-32 attention, and the final stride-4 mask prediction remain unchanged. Each transformation can be applied separately to architectures with the corresponding computation.

3.1 MASK-GUIDED SPARSE CROSS-ATTENTION

In masked cross-attention, the preceding mask prediction, thresholded at 0.5, defines which feature cells each query may read, but the original implementation still computes attention scores over all N_s cells. For each query, we consider cells in a fixed, input-independent hash order determined by the query and stride, retaining up to $M = 128$ cells permitted by its binary mask. We call this hash-uniform selection: the repeatable pseudorandom order samples across the permitted region without a learned router, runtime randomness, or probability ranking. If the mask permits between one and M cells, we retain them all and mask any padded slots so they receive zero attention weight. If no cells are permitted, attention uses M cells from the feature grid without applying the mask. We gather features and positional encodings at the selected cells and reuse the original learned attention projections. For N_h attention heads, this reduces the score tensor from $N_h \times Q \times N_s$ to $N_h \times Q \times M$.

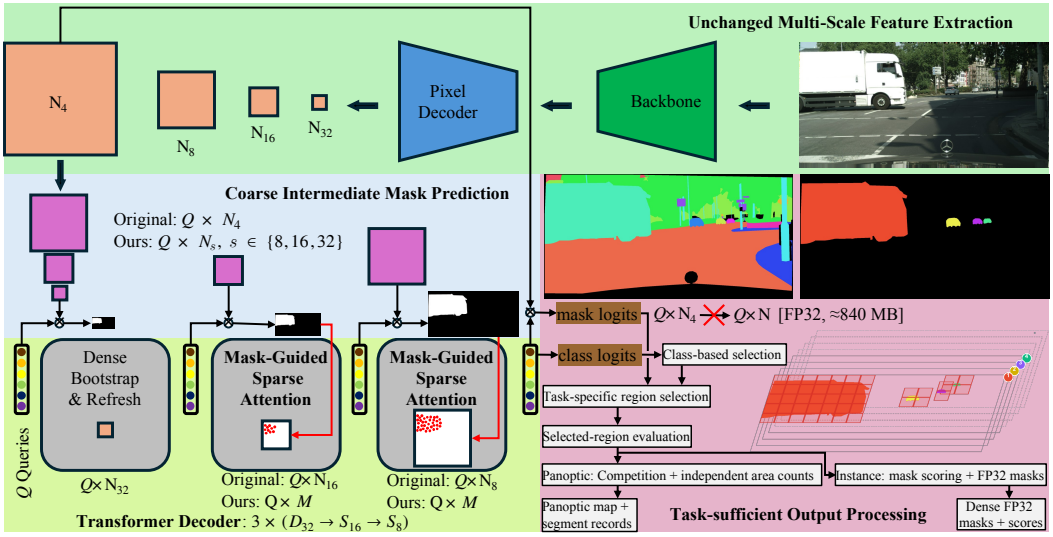


Figure 2: Overview of the pipeline on Mask2Former. Feature extraction and stride-32 attention remain dense. Intermediate masks are predicted at the next attention stride; strides 16 and 8 use M cells per query, with red dots illustrating one query's selection. Highlighted tiles illustrate instance rendering from positive source support for four retained queries; faded planes show other retained queries. Selection uses predicted masks and logits.

Mask2Former's decoder uses feature strides 32, 16, and 8 in repeated groups of three layers. We retain this order as $D_{32} \to S_{16} \to S_8$, with dense ($D$) cross-attention at stride 32 and mask-guided sparse (S) cross-attention at strides 16 and 8. Keeping stride-32 attention dense avoids selection and gathering overhead on its smaller feature grid. Section 4.4 evaluates the token budget $M = 128$ and this attention schedule.

3.2 Coarse Intermediate Mask Prediction

Between decoder layers, Mask2Former predicts Q intermediate masks at stride 4 and immediately downsamples them to the next attention block's stride. Let $E \in \mathbb{R}^{Q \times C}$ be the query mask embeddings, $F_4 \in \mathbb{R}^{C \times N_4}$ the shared mask feature map, and R_s the bilinear resize to stride s. Because resizing is linear over the spatial dimensions,

$$R_s(EF_4) = E\,R_s(F_4) = EF_s, \qquad F_s = R_s(F_4). \tag{1}$$

For each image, we resize the shared stride-4 feature map once per target stride using bilinear interpolation with `align_corners=False`. These features are reused across decoder layers to predict intermediate masks as EF_s. All reported decoder timings with coarse prediction include this per-image resizing. The rewrite is equivalent in real arithmetic. It reduces intermediate logit storage from $Q \times N_4$ to $Q \times N_s$, by up to $64\times$ at stride 32, and avoids repeatedly reading the full stride-4 feature map and writing temporary stride-4 logits. The final mask prediction remains at stride 4 to supply the task output.

3.3 Task-Sufficient Output Processing

The original Mask2Former output path upsamples all query masks before selecting candidates. We apply the unchanged class-based selection first, then restrict mask evaluation using task-specific support and score bounds.

For instance segmentation, masks are processed independently, and candidates sharing a query reuse its mask computation. Because bilinear interpolation uses nonnegative weights, a positive output logit requires a positive source value. We render 128×128 output tiles covering positive source support, including neighboring tiles needed for interpolation. Positive interpolated logits define the

binary instance masks, as in the original implementation. Sigmoid probabilities summed over these positive pixels and the corresponding pixel counts determine the mask-quality scores. The binary tiles are written directly into dense FP32 masks in the original candidate order, avoiding intermediate full-resolution mask copies.

For panoptic segmentation, queries compete at each pixel using class confidence times mask probability. A query whose mask probability is below the threshold at a pixel can still win there, so positive support alone is insufficient. Segment filtering compares the total number of pixels won by each query with its full thresholded mask area, including pixels won by other queries.

For each retained query q and 32×32 output region r, we compute a conservative lower bound L_{qr} and upper bound U_{qr} on its weighted score at every pixel in the region. These bounds use all source logits contributing to interpolation within the region, with the original handling of image boundaries. We exclude a query from competition only when

$$U_{qr} < \max_j L_{jr}. \tag{2}$$

Its best possible score is then below another query's guaranteed score everywhere in the region. The strict inequality retains possible ties. Remaining candidates compete in the original query order.

We account for original mask area independently using probability bounds. A region certified to lie entirely at or above the 0.5 probability threshold contributes all its pixels; one entirely below contributes zero. Ambiguous regions are evaluated even for queries excluded from competition, reusing values when available. The resulting counts and thresholded winners feed the original segment filtering and stuff merging. Rejected pixels remain void, without reassignment.

Panoptic processing constructs the label map and ordered segment records without a dense full-resolution query-mask stack. Semantic output remains dense. We retain the original operators for bilinear interpolation, sigmoid, and elementwise score arithmetic, while adapting reductions and output assembly to the sparse layout. Numerical conditions and output-agreement checks are described in the appendix.

4 Experiments

4.1 Experimental Setup

We evaluate panoptic, instance, and semantic segmentation on Cityscapes (Cordts et al., 2016), COCO (Lin et al., 2014), and ADE20K (Zhou et al., 2017) using PQ, AP, and mIoU where applicable. We use released checkpoints when available and otherwise train models from published implementations or specifications. Deployment measurements use an AGX Orin at batch size one and native Cityscapes resolution (1024×2048), in FP32 unless stated otherwise. Complete-model latency includes all network stages and output processing. We select $M = 128$ and the attention schedule on Cityscapes and keep both fixed for transfer experiments.

4.2 Main Results on Cityscapes

Tables 1 and 2 show that the applicable transformations accelerate inference across the evaluated model families, with reported PQ and AP changes of at most 0.4 points. Gains are larger for Mask2Former variants with lighter pixel decoders, consistent with a larger share of runtime in the stages we optimize. For PEM, coarse mask prediction and task-sufficient output processing alone provide a $1.27\times$ panoptic speedup and a $1.53\times$ instance speedup, with unchanged accuracy. PEM thus offers a favorable accuracy–latency trade-off, showing that already efficient architectures also benefit.

Table 3 provides our primary paired comparison against an optimized dense reference in FP32 and FP16 automatic mixed precision (AMP). Across R50 and R18 models with three, six, or nine decoder layers, the complete pipeline achieves 1.24–$1.35\times$ panoptic speedups and 1.43–$1.59\times$ instance speedups.

Table 1: **Panoptic segmentation on Cityscapes val.** Native 1024×2048 resolution, batch size one, and FP32 on an AGX Orin. Filled dots indicate which of our transformations are applied: mask-guided sparse cross-attention (S), coarse intermediate-mask prediction (C), and task-sufficient output processing (P). Best latency, FPS, and accuracy values are bold; second-best values are underlined. All models use R50 except Panoptic-DL (R52-DC5).

				Latency		Speedup			
Model	S	C	P	(ms) ↓	FPS ↑	↑	PQ ↑	PQ^{Th}	PQ^{St}
Panoptic-DL (Cheng et al., 2020)				411.6	2.43	-	60.3	50.9	67.1
YOSO (Hu et al., 2023)				278.6	3.59	-	59.7	51.0	66.1
kMaX-DL (Yu et al., 2022)				925.1	1.08	-	**63.5**	**57.8**	**67.7**
M2F (Cheng et al., 2022)				467.5	2.14	-	<u>62.1</u>	<u>54.9</u>	67.3
M2F (ours)	•	•	•	371.2	2.69	1.26	<u>62.1</u>	<u>54.9</u>	<u>67.4</u>
PRO-SCALE (Aich et al., 2025)				407.9	2.45	-	62.0	54.8	67.3
PRO-SCALE (ours)	•	•	•	307.6	3.25	1.33	62.0	54.6	<u>67.4</u>
PEM (Cavagnero et al., 2024)				269.6	3.71	-	60.8	53.2	66.3
PEM (ours)		•	•	<u>212.1</u>	<u>4.71</u>	1.27	60.8	53.2	66.3
FastInst (He et al., 2023b)				297.8	3.36	-	54.3	43.2	62.3
FastInst (ours)			•	243.2	4.11	1.22	54.3	43.2	62.3
Compact-M4 (Busemann et al., 2026)				299.9	3.33	-	54.6	45.1	61.4
Compact-M4 (ours)			•	250.5	3.99	1.20	54.6	45.1	61.4
Compact-M8 (Busemann et al., 2026)				238.7	4.19	-	53.4	43.9	60.2
Compact-M8 (ours)			•	**188.9**	**5.29**	1.26	53.4	43.9	60.2
PPM-FPN M2F (He et al., 2023b; Cheng et al., 2022)				323.1	3.10	-	56.0	45.7	63.4
PPM-FPN M2F (ours)	•	•	•	226.7	4.41	1.43	55.7	45.2	63.4
RT-DETR M2F (Zhao et al., 2024; Cheng et al., 2022)				361.2	2.77	-	56.6	48.3	62.7
RT-DETR M2F (ours)	•	•	•	266.0	3.76	1.36	56.5	48.1	62.5

Table 2: **Instance segmentation on Cityscapes val.** Same setting and notation as Table 1. Speedups use the corresponding original baseline.

				Latency		Speedup	
Model	S	C	P	(ms) ↓	FPS ↑	↑	AP ↑
YOSO (Hu et al., 2023)				302.8	3.30	-	33.8
kMaX-DL (Yu et al., 2022)				1098.6	0.91	-	**38.6**
M2F (Cheng et al., 2022)				462.9	2.16	-	<u>37.3</u>
M2F (ours)	•	•	•	353.6	2.83	1.31	37.2
PRO-SCALE (Aich et al., 2025)				430.5	2.32	-	36.9
PRO-SCALE (ours)	•	•	•	286.8	3.49	1.50	36.9
PEM (Cavagnero et al., 2024)				292.0	3.42	-	35.3
PEM (ours)		•	•	<u>191.2</u>	<u>5.23</u>	1.53	35.3
FastInst (He et al., 2023b)				318.1	3.14	-	25.7
FastInst (ours)			•	222.8	4.49	1.43	25.7
Compact-M4 (Busemann et al., 2026)				323.1	3.09	-	25.8
Compact-M4 (ours)			•	231.6	4.32	1.40	25.8
Compact-M8 (Busemann et al., 2026)				261.2	3.83	-	25.7
Compact-M8 (ours)			•	**166.6**	**6.00**	1.57	25.7
PPM-FPN M2F (He et al., 2023b; Cheng et al., 2022)				353.7	2.83	-	27.2
PPM-FPN M2F (ours)	•	•	•	210.5	4.75	1.68	27.2
RT-DETR M2F (Zhao et al., 2024; Cheng et al., 2022)				384.9	2.60	-	27.6
RT-DETR M2F (ours)	•	•	•	244.4	4.09	1.57	27.2

Table 3: **Paired full-model comparison with an optimized dense reference.** M2F with a PPM-FPN pixel decoder on an AGX Orin at 1024×2048. Output processing uses FP32 for both methods. Latencies report mean $\pm$ sample SD across five process medians.

Task	Backbone	Decoder layers	Precision	Optimized dense (ms)	Ours (ms)	Speedup	PQ/AP dense / ours
Panoptic	R50	9	FP32	296.5 ± 0.9	228.2 ± 1.0	$\mathbf{1.30}\times$	56.0 / 55.7
	R50	9	AMP	254.0 ± 0.5	195.5 ± 1.6	$\mathbf{1.30}\times$	55.9 / 55.8
	R18	6	FP32	196.2 ± 0.4	145.3 ± 0.5	$\mathbf{1.35}\times$	53.1 / 53.2
	R18	6	AMP	183.7 ± 0.8	140.7 ± 0.7	$\mathbf{1.31}\times$	53.1 / 53.2
	R18	3	FP32	171.7 ± 0.4	135.1 ± 0.7	$\mathbf{1.27}\times$	51.6 / 51.7
	R18	3	AMP	159.1 ± 0.4	127.9 ± 0.4	$\mathbf{1.24}\times$	51.5 / 51.4
Instance	R50	9	FP32	297.3 ± 0.8	208.2 ± 0.5	$\mathbf{1.43}\times$	27.2 / 27.2
	R50	9	AMP	254.8 ± 0.9	176.8 ± 0.9	$\mathbf{1.44}\times$	27.2 / 27.2
	R18	6	FP32	200.0 ± 3.0	125.8 ± 0.4	$\mathbf{1.59}\times$	22.3 / 22.8
	R18	6	AMP	185.6 ± 0.3	120.9 ± 0.5	$\mathbf{1.54}\times$	22.3 / 22.7
	R18	3	FP32	174.1 ± 0.6	115.4 ± 0.3	$\mathbf{1.51}\times$	19.9 / 19.7
	R18	3	AMP	161.8 ± 1.0	108.6 ± 0.3	$\mathbf{1.49}\times$	19.8 / 19.8

4.3 Transfer Across Datasets, Tasks, and Architectures

We evaluate mask-guided sparse attention with the Cityscapes-selected budget of $M = 128$ on COCO (Table 4) and ADE20K (Appendix, Table 8). Across Mask2Former, OneFormer, PRO-SCALE, and MP-Former, accuracy changes relative to dense execution remain within ± 0.4 points for R50 and ± 0.1 for Swin-L. The tested conversion of FastInst's dual-path decoder loses 1.2 AP, with a 2.2-point drop on large objects.

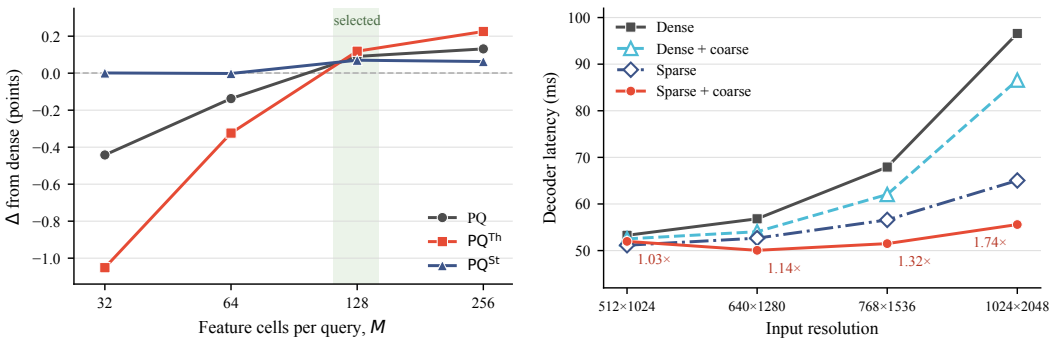
Table 4: **Transfer to COCO val2017.** Cells report dense / ours ($M = 128$).

Model	Backbone	PQ	AP	mIoU
M2F (pan.) (Cheng et al., 2022)	R50	51.9 / 51.9	41.7 / 41.7	61.7 / 61.5
M2F (inst.)	R50	-	43.7 / 43.7	-
PRO-SCALE (pan.) (Aich et al., 2025)	R50	51.4 / 51.4	41.4 / 41.4	61.6 / 61.5
MP-Former (inst.) (Zhang et al., 2023)	R50	-	40.2 / 40.1	-
FastInst (inst.) (He et al., 2023b)	R50	-	37.9 / 36.7	-
M2F (pan.)	Swin-L	57.8 / 57.9	48.6 / 48.7	67.4 / 67.4
OneFormer (univ.) (Jain et al., 2023)	Swin-L	57.9 / 57.9	49.0 / 49.0	67.2 / 67.1

4.4 Mechanism Ablations

Selected feature cells per query. Figure 3(a) shows that stuff-class PQ remains at the dense level even with only $M = 32$ feature cells per query, suggesting that the large regions these classes often occupy tolerate aggressive subsampling. Thing-class accuracy decreases with smaller budgets. A budget of $M = 128$ preserves dense-level PQ, with little further improvement at $M = 256$. We therefore use $M = 128$ by default.

Input resolution. Figure 3(b) shows that combining mask-guided sparse attention and coarse mask prediction gives little decoder speedup at 512×1024 ($1.03\times$), rising to $1.74\times$ at 1024×2048. Higher resolutions increase the number of feature cells processed by dense attention, while the sparse budget remains $M = 128$ per query. At native Cityscapes resolution (1024×2048), hardware counters on an RTX A4000 show that these two transformations reduce decoder DRAM traffic from 12.8 to 4.2 GB per image (67%) in FP32. Across the tested resolutions, PQ differences from dense execution range from -0.05 to $+0.19$ points.



(a) Per-query attention budget (b) Input resolution

Figure 3: **Decoder mechanism ablations.** M2F R50 on Cityscapes val. (a) Change in PQ and its thing/stuff components relative to dense execution as the number of selected feature cells per query varies. (b) Decoder latency across input resolutions on an AGX Orin. Labels show the combined speedup of mask-guided sparse attention and coarse mask prediction over the original dense decoder.

Dense stride-32 attention. Table 5(a) shows that sparsifying all three stride-32 blocks reduces PQ by 0.5 points, mainly on thing classes. Keeping the first block dense restores accuracy. Keeping all three dense yields similar accuracy and Orin decoder latency to keeping only the first dense (64.5 versus 63.5 ms, with coarse mask prediction disabled).

Selection strategy. Table 5(b) compares selection strategies at $M = 128$. Selecting feature cells with the highest mask probabilities reduces PQ, mainly for thing classes. Including cells with probabilities just above 0.5 recovers most of this loss. Hash-uniform selection achieves the highest PQ among the tested strategies while avoiding input-dependent ranking, and we use it by default.

Table 5: **Decoder design ablations.** Cityscapes panoptic validation with $M = 128$.

(a) Dense stride-32 schedule

Setting	Dense blocks	PQ	PQ^{Th}	PQ^{St}
All sparse	none	61.6	53.5	67.4
Dense bootstrap	1	62.2	55.0	67.4
+ refresh	1, 4, 7	62.1	54.9	67.4

(b) Selection strategy

Strategy	PQ	PQ^{Th}	PQ^{St}
Hash-uniform	62.1	54.9	67.4
Top-confidence	61.5	53.8	67.1
+ near-threshold	62.0	55.0	67.1

Sparse training and test transfer. Without retraining, the released checkpoint retains 60.0 PQ on the held-out test set using the validation-selected hyperparameters unchanged (Table 6). Sparse training offers no consistent gain over dense training in our matched single-seed comparison.

Table 6: **Sparse training vs. train-free execution.** Cityscapes panoptic accuracy on val and test.

	Validation			Test		
Setting	PQ	PQ^{Th}	PQ^{St}	PQ	PQ^{Th}	PQ^{St}
Released, dense inference	62.1	54.9	67.3	60.0	50.6	66.8
Released, sparse inference	62.1	54.9	67.4	60.0	50.6	66.8
Local, dense-trained	62.6	55.5	67.7	60.2	51.3	66.7
Local, sparse-trained	62.7	55.4	68.0	60.1	50.8	66.9

Optimized dense attention. Against PyTorch's fused dense SDPA, mask-guided sparse attention retains a $1.51\times$ decoder speedup, with $3.22\times$ faster attention at stride 8 (Appendix, Table 10).

4.5 Output Processing Efficiency

Table 7 compares original output processing with progressively optimized dense variants and our sparse method. The original implementation interpolates all query masks before selection; early class filtering avoids processing discarded queries. Removing redundant resizing reduces panoptic latency from 64.9 to 60.7 ms. For instance segmentation, this change together with vectorized filtering and reuse of masks shared by multiple candidates reduces latency from 84.9 to 60.9 ms. All variants retain the original operators for bilinear interpolation, sigmoid, and elementwise score arithmetic; sparse processing adapts reductions and output assembly to its selected regions.

Table 7: **Output processing efficiency.** M2F R50, FP32, 1024×2048. Output latency is measured on an AGX Orin. Additional optimizations remove redundant resizing and, for instances, vectorize filtering and reuse query masks.

Task	Implementation	Orin latency (ms)	A4000 DRAM MB/image
Panoptic	Original	88.7	9815.7
	+ Early class filtering	64.9	7574.5
	+ Additional optimizations	60.7	7574.5
	Sparse (ours)	**34.0**	**1142.8**
Instance	Original	107.7	10051.7
	+ Early class filtering	84.9	7046.3
	+ Additional optimizations	60.9	8147.3
	Sparse (ours)	**14.4**	**1584.8**

Against these optimized dense baselines, our method reduces panoptic latency to 34.0 ms, saving 26.7 ms and providing a $1.8\times$ speedup. Instance latency falls to 14.4 ms, saving 46.5 ms and providing a $4.2\times$ speedup. A4000 DRAM traffic decreases by 84.9% for panoptic and 80.5% for instance processing. Separate audits using identical input logits confirmed exact agreement with original outputs on all 500 Cityscapes validation images for panoptic maps, instance masks, and labels. Instance masks permit independent pruning based on positive support, whereas panoptic processing must retain possible competitors and count mask areas independently of pixel winners.

5 Conclusion

Frozen segmentation checkpoints already produce masks and logits that enable substantial reductions in inference cost without retraining. Mask-guided sparse attention with 128 feature cells per query preserves dense-level accuracy across compatible checkpoints, datasets, and tasks using unchanged settings. We develop task-sufficient output processing for instance and panoptic segmentation, avoiding mask evaluation in regions that cannot affect the final output. For panoptic segmentation, we find that masks can influence pixel assignments even where they predict background, and derive conditions for safely skipping computation. Combined with coarse intermediate mask prediction, these transformations reduce decoder DRAM traffic by 67% versus original execution and output-processing traffic by over 80% versus optimized dense processing. Complete-model inference on an AGX Orin achieves speedups of up to $1.35\times$ for panoptic and $1.59\times$ for instance segmentation over optimized dense references, with similar gains under mixed precision.

The benefits depend on image resolution and decoder architecture. Decoder gains are small at low resolution, and the tested FastInst sparse conversion loses accuracy. Output-stage speedups are smaller for panoptic than instance segmentation, while semantic output remains dense. Our measurements cover the evaluated PyTorch/CUDA implementations; validated end-to-end TensorRT integration and evaluation remain future work.

Reproducibility Statement

We will share an anonymized implementation for panoptic and instance segmentation with reviewers and area chairs through a private OpenReview comment when discussion opens. The code will be released publicly upon acceptance. Experimental settings are described in Section 4, with numerical validation in Appendix A and hardware settings and timing protocols in Appendix C.

AI Use Statement

We used AI tools to improve the grammar, wording, clarity, and conciseness of author-provided manuscript text and to identify additional related work. AI tools also assisted with implementing author-specified ideas, executing code, and generating and refining figures according to author instructions, excluding the method overview. AI tools also helped develop, refine, and write the panoptic correctness and numerical argument and provided feedback on author-designed experimental comparisons and the interpretation of their results. The authors take responsibility for the final manuscript, implementation, figures, and scientific claims.

References

Abhishek Aich, Yumin Suh, Samuel Schulter, and Manmohan Chandraker. Progressive token length scaling in transformer encoders for efficient universal segmentation. In *International Conference on Learning Representations*, 2025.

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

Oliver Busemann, Torben Töniges, and Martin Atzmueller. Memory bandwidth, not FLOPs: Profiling and accelerating query-based segmentation decoders on edge GPUs. In *British Machine Vision Conference*, 2026.

Niccolò Cavagnero, Gabriele Rosi, Claudia Cuttano, Francesca Pistilli, Marco Ciccone, Giuseppe Averta, and Fabio Cermelli. PEM: Prototype-based efficient MaskFormer for image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Bowen Cheng, Maxwell D. Collins, Yukun Zhu, Ting Liu, Thomas S. Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-DeepLab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. In *Advances in Neural Information Processing Systems*, 2021.

Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 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 *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

Calvin Galagain, Martyna Poreba, and François Goulette. Token-space mask prediction for efficient vision transformer segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2026.

Haoyu He, Jianfei Cai, Zizheng Pan, Jing Liu, Jing Zhang, Dacheng Tao, and Bohan Zhuang. Dynamic focus-aware positional queries for semantic segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023a.

Junjie He, Pengyu Li, Yifeng Geng, and Xuansong Xie. FastInst: A simple query-based model for real-time instance segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023b.

Jie Hu, Linyan Huang, Tianhe Ren, Shengchuan Zhang, Rongrong Ji, and Liujuan Cao. You only segment once: Towards real-time panoptic segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Jitesh Jain, Jiachen Li, Mang Tik Chiu, Ali Hassani, Nikita Orlov, and Humphrey Shi. OneFormer: One transformer to rule universal image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Tommie Kerssies, Niccolò Cavagnero, Alexander Hermans, Narges Norouzi, Giuseppe Averta, Bastian Leibe, Gijs Dubbelman, and Daan de Geus. Your ViT is secretly an image segmentation model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image segmentation as rendering. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

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 *Advances in Neural Information Processing Systems*, 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*, 2014.

Haisong Liu, Yang Chen, Haiguang Wang, Zetong Yang, Tianyu Li, Jia Zeng, Li Chen, Hongyang Li, and Limin Wang. Fully sparse 3D occupancy prediction. In *European Conference on Computer Vision*, 2024a.

Zhijian Liu, Zhuoyang Zhang, Samir Khaki, Shang Yang, Haotian Tang, Chenfeng Xu, Kurt Keutzer, and Song Han. Sparse refinement for efficient high-resolution semantic segmentation. In *European Conference on Computer Vision*, 2024b.

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

Qihang Yu, Huiyu Wang, Siyuan Qiao, Maxwell Collins, Yukun Zhu, Hartwig Adam, Alan Yuille, and Liang-Chieh Chen. k-means mask transformer. In *European Conference on Computer Vision*, 2022.

Hao Zhang, Feng Li, Huaizhe Xu, Shijia Huang, Shilong Liu, Lionel M. Ni, and Lei Zhang. MP-Former: Mask-piloted transformer for image segmentation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Jintao Zhang, Chendong Xiang, Haofeng Huang, Jia Wei, Haocheng Xi, Jun Zhu, and Jianfei Chen. SpargeAttention: Accurate and training-free sparse attention accelerating any model inference. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 76397–76413, 2025.

Yian Zhao, Wenyu Lv, Shangliang Xu, Jinman Wei, Guanzhong Wang, Qingqing Dang, Yi Liu, and Jie Chen. DETRs beat YOLOs on real-time object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ADE20K dataset. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable DETR: Deformable transformers for end-to-end object detection. In *International Conference on Learning Representations*, 2021.

A CORRECTNESS AND NUMERICAL SCOPE

We state the assumptions, preservation arguments, and numerical limitations for sparse attention, coarse prediction, and task-specific output processing.

A.1 SPARSE CROSS-ATTENTION SUPPORT

On the target attention grid, cells with sigmoid mask probability at least 0.5 are allowed. We retain up to M distinct allowed cells in fixed hash order, masking any padding. When between one and M cells are allowed, all are retained, preserving the dense masked-attention operation in real arithmetic for identical inputs. Floating-point results are not necessarily bit-identical. If no cells are allowed, sparse blocks attend to the selected M cells without masking, while dense blocks clear the mask and attend to the full feature grid. Under the default schedule, stride 32 remains dense and strides 16 and 8 are sparse. This fixed-budget operation approximates dense attention; accuracy is evaluated empirically, not guaranteed by support selection.

A.2 COARSE INTERMEDIATE PREDICTION

Both computation orders use the same resizing settings, but floating-point rounding can change mask decisions near the threshold. Across two models in our FP32 setting, each evaluated on all 500 Cityscapes validation images, three such decisions changed. Selected tokens and final outputs remained unchanged in both sparse configurations. The largest mask-score difference in these configurations was 5.19×10^{-4}.

A.3 INSTANCE OUTPUT PRESERVATION

Given identical class and final mask logits and unchanged candidate selection, the interpolation halo adds neighboring tiles, including diagonals, to cover positive pixels that interpolation can produce across tile boundaries. In the existing audit on all 500 Cityscapes validation images, tiled processing matched original masks, labels, and candidate order exactly, with a maximum score difference of 1.67×10^{-6}; direct copying into dense FP32 masks preserves these outputs. Instance masks use the strict logit threshold $z > 0$, while panoptic masks use the probability threshold $\sigma(z) \geq 0.5$, matching the original implementation.

A.4 PANOPTIC OUTPUT PRESERVATION

Given identical class and final mask logits and unchanged candidate filtering, the score limits exclude only queries that score below another query throughout a region. They therefore cannot win. Retaining possible ties and the original query order preserves winner selection. Counting original mask areas independently also includes pixels won by other queries. With unchanged threshold decisions and assembly rules, this preserves the filtering areas, panoptic map, and ordered segment records.

Floating-point rounding must not cause us to discard a query that could win a pixel. We therefore construct score limits that cover both the mathematical range and numerical rounding. For each region, we take the source-logit extrema a, b over all pixels contributing to interpolation, including clamped image boundaries. For the tested fourfold and eightfold bilinear upsampling, coordinates and weights are exactly representable. For finite FP32 inputs with $A = \max(|a|, |b|) \leq 2^{120}$, we expand the range to $[a - E, b + E]$, where

$$E = 32uA + 64\eta, \qquad u = 2^{-24}, \quad \eta = 2^{-126}.$$

This allowance exceeds the rounding bound of $10uA$ for at most nine arithmetic operations, plus 40η to cover the numerical error when extremely small values become zero. We compute the endpoints in FP64, then round the lower endpoint downward and the upper endpoint upward when converting to FP32, ensuring that conversion cannot shrink the range.

A lookup table evaluates the actual sigmoid kernel for every finite FP32 input in all four execution positions. It supplies limits covering every sigmoid output within the expanded range, including numerical irregularities. After multiplication by class confidence, we again round the lower limit

downward and the upper limit upward. These guarantees apply only to the matching GPU, software binaries, and kernel configuration. Under these conditions, a query is skipped only in regions where it cannot win a pixel.

We checked renderer agreement separately using identical input logits: panoptic maps and ordered segment records matched original output exactly on all 500 Cityscapes validation images. Changes to precision, hardware, software, or interpolation settings require revalidation.

B Additional Experimental Results

We provide supplementary result tables and ablations omitted from the main text.

B.1 Complete Dataset and Task Tables

Table 8: **Transfer to ADE20K val.** Cells report dense / ours ($M = 128$).

Model	Backbone	PQ	AP	mIoU
M2F (pan.) (Cheng et al., 2022)	R50	39.6 / 39.2	26.4 / 26.4	45.8 / 46.2
M2F (inst.)	R50	-	26.4 / 26.4	-
M2F (sem.)	R50	-	-	47.3 / 47.1
OneFormer (univ.) (Jain et al., 2023)	Swin-L	50.0 / 49.9	35.9 / 35.9	56.9 / 56.9

B.2 Selector and Budget Ablations

Selection order. On the frozen Mask2Former R50 Cityscapes panoptic checkpoint, three predetermined hash phases yield 61.9–62.1 PQ (mean 62.0), compared with 62.1 for dense execution. Only the deterministic selection order changes; $M = 128$ and the decoder schedule remain fixed. The default phase was fixed before this comparison. These results suggest that dense-level accuracy does not depend on one particular selection order in this setting.

Per-stride budgets. On the same checkpoint, reducing the stride-16 budget to 32 lowers PQ by 0.12 points relative to uniform $M = 128$, whereas reducing the stride-8 budget to 32 lowers PQ by 0.46 points. The smaller stride-16 budget accelerates that attention operation by $2.08\times$ on an AGX Orin, but reduces decoder latency by only 0.5% and panoptic inference latency by 0.1%. These small gains are within measurement uncertainty, so we retain $M = 128$ at both strides.

Allowed feature cells. On all 500 Cityscapes validation images, we count the feature cells that each query's predicted mask permits attention to use at strides 16 and 8. We group queries by their final predicted thing/stuff class after class filtering and pool measurements across the three attention layers at each stride.

Table 9: **Feature cells permitted by predicted masks.** Mask2Former R50 with coarse mask prediction, evaluated separately at each budget. "All cells selected" excludes empty masks, which require fallback attention. Median counts include empty masks.

Budget M	Stride	Query type	Median allowed cells	All cells selected (%)	Empty masks (%)
128	16	Thing	4	93.7	20.6
128	16	Stuff	174	42.7	5.1
128	8	Thing	16	80.7	16.9
128	8	Stuff	713	20.4	4.1
32	16	Thing	4	80.1	18.5
32	16	Stuff	174	20.5	5.2
32	8	Thing	18	55.7	15.3
32	8	Stuff	713	7.8	4.0

With $M = 128$, all permitted cells are selected for most nonempty thing masks. Stuff masks typically allow attention over much larger regions: at stride 8, the median is 713 cells. With $M = 32$, only 7.8% of nonempty stuff masks fit within the budget; the remaining 92.2% are subsampled. Together with the preserved stuff PQ in the budget ablation (Figure 3(a)), this shows that substantial subsampling of these regions is possible without reducing aggregate stuff accuracy in this setting. The original dense execution shows similar mask-size distributions.

Table 10: **Attention execution by stride.** M2F R50 at 1024×2048, batch size one, FP32 on an AGX Orin, with coarse mask prediction enabled throughout. Stride timings are measured separately from the complete decoder.

		Cross-attention latency (ms)			Decoder latency
Sparse strides	Dense backend	Stride 32	Stride 16	Stride 8	(ms)
None	Original MHA	5.22	8.64	33.98	86.56
None	Fused SDPA	4.00	8.22	32.77	82.59
16, 8	Original MHA	5.41	9.48	10.15	55.44
16, 8	Fused SDPA	4.25	9.54	10.18	54.57
8	Fused SDPA	4.17	8.23	10.21	55.10

C Reproducibility Details

Deployment requires kernel compilation and, for panoptic score bounds, runtime-specific numerical certification, without model training or labeled calibration. Reported inference latency excludes this setup and warmup but includes per-image selection and output processing.

Unless otherwise stated, each configuration is measured in five fresh processes, with 50 warmup runs and 200 timed validation images per process; latencies are summarized from per-process medians. Complete-model latency uses synchronized wall timing and includes the backbone, pixel decoder, transformer decoder, prediction heads, and output processing. GPU component timings use CUDA events. Instance output timings include dense FP32 binary-mask construction. Evaluator serialization is excluded.

Tables 1 and 2 report median latency from one process with 50 warmups and all 500 validation images; speedups are ratios of medians. For Table 3, the dense reference uses fused dense SDPA, original intermediate-mask prediction, and the additional output optimizations in Table 7. AMP includes the PPM-FPN pixel decoder; both output paths use FP32, with conversion included in complete-model latency.

Decoder latencies in Figure 3(b), Table 10, and the stride-32 schedule comparison are medians over 200 images in one process after 50 warmups. Sparse calls use fused SDPA. The FP32 SDPA measurements in Table 10 use PyTorch's memory-efficient backend. Stride columns sum three attention calls, including selection and gathering but excluding mask clearing for empty masks in dense blocks. Decoder totals include this mask clearing, all nine blocks, coarse-feature resizing, and predictions, excluding the backbone and pixel decoder.

Table 7 reports median output latency from one process using cached Cityscapes logits; DRAM traffic is averaged over ten inputs.

Measurements were run without competing GPU workloads, with clock and temperature monitoring.

Table 11: AGX Orin hardware and measurement settings.

Setting	Value
Device / memory	NVIDIA Jetson AGX Orin Developer Kit, 64 GB shared memory
Software	PyTorch 2.4 (NVIDIA build), CUDA 12.2
Precision / TF32	FP32; TF32: matmul on, cuDNN on
Power mode	MAXN
Clocks	CPU / GPU / memory: 2.2 / 1.3 / 3.2 GHz

D QUALITATIVE RESULTS

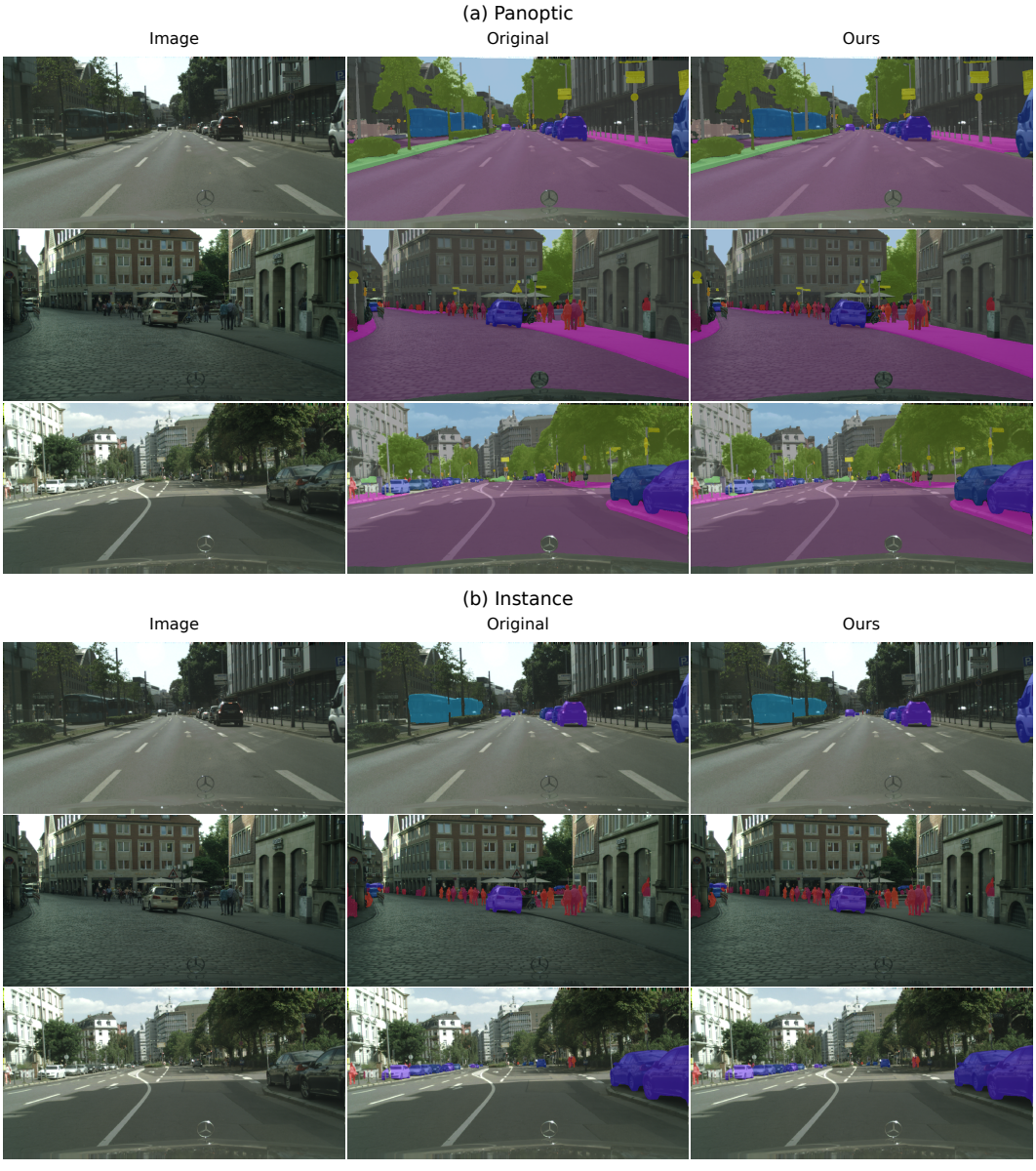


Figure 4: Left: Cityscapes validation images. Middle: predictions from frozen Mask2Former checkpoints. Right: our methods applied to the same checkpoints.