Local Non-Maximum Suppression: Enhancing Object Detection in Large-Scale Remote Sensing Images via iterative pipelined Postprocessing

Bettina Felten[1], Wolfgang Gross[1], Andreas Michel[1]

[1] Fraunhofer IOSB, Gutleuthausstr. 1, 76275 Ettlingen, Germany
- (bettina.felten, wolfgang.gross, andreas.michel)@iosb.fraunhofer.de

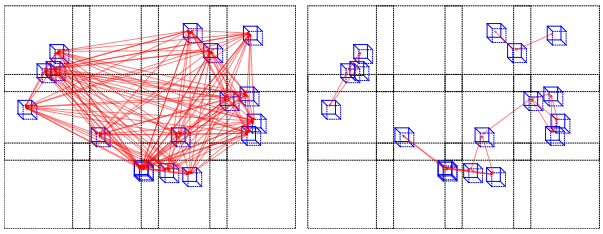
Keywords: Slice-wise inference, Non-Maximum Suppression, Large-scale imagery, Dense scenes, Tiny Object Detection, Oriented Object Detection.

Abstract

Object detection in large, dense remote sensing imagery is difficult because targets are often small and arbitrarily oriented, and state-of-the-art detectors cannot process very large images directly without a reduction in accuracy. Tiling-based inference workflows mitigate the latter issue by running inference iteratively on overlapping tiles, but introduce pre- and postprocessing overhead for image tiling and Non-Maximum Suppression (NMS). We introduce local NMS, an asynchronous tile-wise postprocessing scheme. Local NMS runs in a separate subprocess in parallel to tile-wise inference and collects intermediate results enqueued by the inference process, immediately applying postprocessing. Intelligent reordering of tiles in a preprocessing step ensures optimal usage of computing resources. We assess our method using three state-of-the art object detection models for horizontal and oriented bounding box detection on two benchmark datasets containing large dense aerial and satellite images, DOTA-v2.0 and Izembek Lagoon Birds, stratifying by image size and average object density. Local NMS consistently reduces end-to-end runtime across models and datasets without significant impact on mAP. A maximum runtime reduction of 60.77% on large dense DOTA-v2.0 scenes could be achieved without modifying model architectures or retraining.

1. Introduction

Object detection in remote-sensing images presents unique challenges, including arbitrary orientations, scale and density variations (Ding et al., 2021b), and the prevalence of small and tiny objects, as well as very large images with up to several hundred million pixels (Michel et al., 2020, Akyon et al., 2022). Detecting small objects in these images is extremely relevant for a wide range of applications such as traffic and wildlife monitoring, reconnaissance, precision agriculture or disaster response (Hua and Chen, 2025). Tiling approaches as implemented in the Slicing-Aided Hyper Inference (SAHI) Python framework (Akyon et al., 2022) improve small-object detection in large images by processing overlapping tiles during inference rather than resizing large scenes. Resizing significantly reduces the quality of object detection results due to information loss and changed aspect ratios, particularly for small objects (Michel et al., 2022). An overlap between tiles is required to avoid incomplete or missing detections due to objects being divided between tiles. Tiling-based approaches necessarily introduce processing overhead for image tiling as well as postprocessing; due to the overlapping regions being passed to the inference model multiple times, inference often produces redundant duplicate or partial bounding boxes for objects located in overlapping areas. These duplicate objects need to be eliminated with NMS, an efficient method to suppress overlapping bounding boxes by removing detections which have significant overlap with another bounding box with a higher confidence score. NMS is often part of object detection inference pipelines to improve output quality. However, each additional superfluous bounding box adds a quadratic increase of pairwise operations, as illustrated in Figure 1(a). Additionally, coordinates of detected objects are transformed from tile coordinates to image coordinates by adding an offset value. Many real-world use cases require very fast, near real-time, or real-time inference.



(a) Global NMS runs pair-wise comparisons on all detected objects.

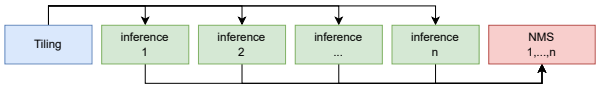
(b) Local NMS only compares objects to objects located in tiles they overlap with.

Figure 1. Number of pair-wise operations of local vs. global NMS on tiled scenes.

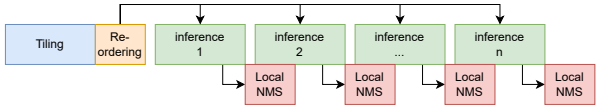
While efforts to improve model inference performance often focus on model architecture design (Liu et al., 2024, Tong et al., 2024), possibilities to optimize the inference workflow as a whole remain widely under-explored.

In existing tile-wise inference approaches such as SAHI (Akyon et al., 2022), postprocessing is performed either globally after collecting all detections or locally within a sliding buffer of consecutive tiles to reduce random access memory consumption. In both cases, inference and postprocessing are executed sequentially. Figure 2(a) shows a typical workflow of sequential tile-wise inference, with a preprocessing step for image tiling and a global postprocessing step after inference has concluded. Additionally, in the case of global NMS, all bounding boxes detected in the scene are compared to each other, even when their location on separate non-overlapping tiles makes it impossible for them to overlap, as shown by Figure 1(a). This leads to inefficient use of computing resources, lack of scalability to very large images with dense objects, and thus high overall runtime which increases at least linearly to input size (Akyon et al., 2022), but is also influenced by object density. We address this

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.



(a) Typical workflow of tile-based sequential object detection; image tiling followed by tile-wise inference and finally combining results in a postprocessing step.



(b) Pipelined workflow; local NMS checks if any tile’s complete neighbourhood has been received whenever a new entry is read from the queue, then runs postprocessing on any such tile and removes it from the buffer.

Figure 2. Comparison of standard and pipelined tile-wise inference and postprocessing workflows.

with an iterative, pipelined approach that runs tile-wise local NMS in parallel with inference. Our method collects detections at runtime and iteratively processes them, only comparing objects located in the same or overlapping tiles, as illustrated by Figure 1(b).

In summary, the main contributions of this work are as follows: Our method (i) intelligently reorders tiles, (ii) enqueues inference outputs, and (iii) performs local NMS in a separate Python sub-process to ensure true parallel processing on multi-core CPU systems. Our approach integrates seamlessly into any existing tiling-based object detection inference framework without the need for retraining.

1.1 Related Work

In this section, we review tile-wise inference, object detection, and NMS, the core components of our method.

1.1.1 Tiling-based Inference For effective object detection in large remote sensing scenes, images are typically divided into tiles to preserve spatial resolution and aspect ratio during inference (Akyon et al., 2022, Michel et al., 2022). SAHI is an open-source, architecture-agnostic Python-based toolkit that provides slicing and merging for tile-wise fine-tuning and inference (Akyon et al., 2022). For training, SAHI generates high-resolution aerial crops from benchmarks such as ImageNet (Deng et al., 2009) and MS COCO (Lin et al., 2014), propagating or clipping cross-tile annotations via area-based thresholds. This consistently outperforms naïve full-image resizing, especially for small and dense objects (Akyon et al., 2022). At inference, overlapping windows are resized to the detector’s input size, passed through the model, and reprojected to the original coordinates. An optional full-image pass recovers very large objects. Tile predictions are then reconciled with class-aware NMS to suppress duplicates and border artifacts (Akyon et al., 2022).

Because SAHI processes all tiles uniformly, adjacent tiles often vary in object density and class presence. Thus, tile size and overlap are key hyperparameters. Too little overlap increases boundary truncation and missed detections, whereas too much overlap increases duplicate predictions, memory, and latency. In practice, the overlap should exceed the expected object size to retain border context while controlling overhead.

1.1.2 Object Detection Early object detection relied on template matching and hand-crafted features. Deep-learning-based object detection, encompassing convolutional neural network (CNN) two-stage and one-stage detectors as well as recent transformer methods, has benefited from large-scale datasets and pretraining (Girshick et al., 2014, Ren et al., 2015, Redmon et al., 2016, Liu et al., 2016). Key advances like feature pyramids improved multi-scale recognition of small objects (Lin et al., 2017). We focus on recent models relevant to remote sensing.

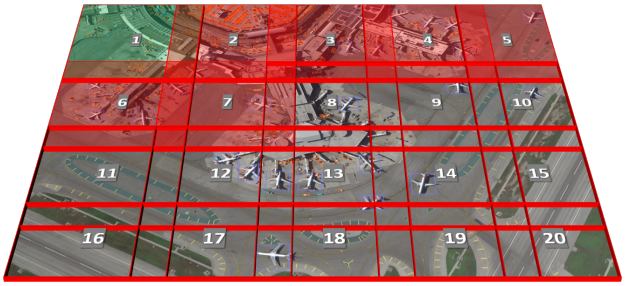
Detection Transformer (DETR) (Carion et al., 2020) introduced a transformer encoder–decoder with set-based, one-to-one label assignment and query-based object representation, offering non-local computation and end-to-end training (Carion et al., 2020, Zhao et al., 2024). DINO (Zhang et al., 2022) improves DETR-style models via denoising during training, anchor selection from encoder positional queries, and gradient-guided weight updates, accelerating training and boosting performance. DINO variants perform well on remote-sensing datasets (Li et al., 2023, Guo and Zhang, 2023).

A key challenge for object detection in remote sensing is arbitrary object orientation, where horizontal bounding boxes (HBB) are suboptimal. Benchmarks such as DOTA (Xia et al., 2018) and FAIR1M (Sun et al., 2022) emphasize oriented bounding boxes (OBB) and evaluation tailored to aerial scenes. Orientation awareness can be introduced via stronger augmentation, alternative box encodings, tailored losses, or explicit group-equivariance that encodes symmetry without increasing parameters (Cohen and Welling, 2016, Wu et al., 2025).

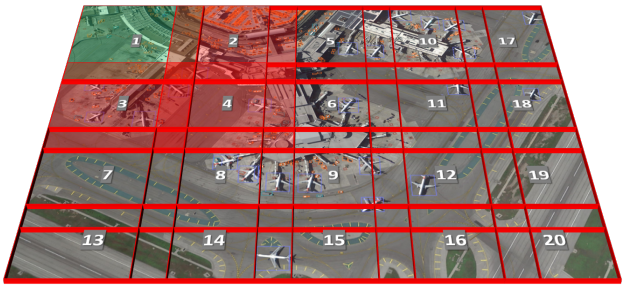
Specialized transformer models leverage global context and oriented queries: OrientedFormer (Zhao et al., 2024) introduces orientation-aware transformer detection through novel techniques such as angle-aware positional encoding/attention, oriented query design, and oriented intersection over union (IoU) matching that enable end-to-end learning of rotated boxes without hand-crafted stages. The Multi-branch head rotation-equivariant single-stage Detector (MessDet) (Wu et al., 2025), another novel oriented object detection architecture, achieves state-of-the-art results with very few parameters using a rotation-equivariant backbone and neck, a new downsampling scheme, a rotation-invariant channel attention mechanism, and group-similarity priors.

1.1.3 NMS Non-Maximum Suppression reduces redundant object proposals in object detection by iterative pair-wise comparisons between boxes based on their overlaps and confidence scores, usually separately for each class. Boxes are first sorted in descending order according to their scores. Then, the algorithm iteratively pops the bounding box with the highest score from the sorted list and compares it to all remaining bounding boxes. Any remaining bounding box with which it has an overlap that exceeds a fixed threshold is eligible for removal. In soft NMS (Bodla et al., 2017), scores are decayed. In hard (greedy) NMS, highly overlapping boxes are removed (Michel, 2025). Variations for OBB and polygons exist. Although greedy NMS’s fixed threshold can introduce some errors, it is still an effective and commonly used technique in object detection workflows (Michel et al., 2022). Naïve NMS scales as $O(n^2)$ with the number of boxes, as illustrated by Figure 1(a). However, each operation is small and highly optimized vectorized implementations are available in toolkits such as MMCV (MMCV Contributors, 2018).

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.



(a) Row-wise processing.



(b) Reordered processing: Neighbours of previous tiles are added first.

Figure 3. Visualisation of row-wise vs. reordered processing. The tiles tinted in red need to be received by the postprocessor before the top left tile (tinted green) can be processed.

2. Methods

Classic tile-based object detection inference workflows consist of three steps, as described in Section 1.1.1 and Figure 2(a): tiling, inference, and NMS. Our iterative pipelined postprocessing starts local NMS on completed tiles while inference continues on remaining tiles, as shown in Figure 2(b). For local NMS to have the same quality as standard global NMS, detection results from all overlapping areas in neighbouring tiles must be considered. Under sliding-window tiling, each tile typically overlaps multiple neighbours, both horizontally and vertically. Thus, row-wise tiling and processing is ill-suited to pipelining. We propose an approach that reorders tiles so that each tile is followed by its neighbours, as described in Section 2.1. Pending detections and metadata are stored in a buffer, which is evaluated by a consumer at each step to identify results that are ready for processing with local NMS. The method is introduced in detail in Section 2.2.

2.1 Reordering Module

We extend the tiling step of tile-wise inference by Algorithm 1, a reordering algorithm which identifies each tile’s overlapping neighbours and reorders execution so that each tile is followed by its neighbours, i.e. the graph of connected tiles is traversed breadth-first, as illustrated in Figure 3. This ensures that postprocessing begins as early as possible and runs continuously throughout inference, minimizing idle time spent waiting for tiles to become available. Simultaneously, it prevents excessive accumulation of predictions in the inter-process queue, thereby reducing the number of detections that must be processed after inference concludes.

The reordering function takes a list of tiles in row-wise or arbitrary order, each mapped to its respective neighbours, as input and returns a reordered copy of the list in traversal order. The algorithm initializes by taking the first element, generally the top left tile, off the list and adding it to the output list. Its

Algorithm 1 Pseudocode for tile reordering

Require: T: List of tile IDs
Require: N: Map of tile IDs to their neighbours
Ensure: O: list of tile IDs in traversal order
$O \leftarrow \langle\rangle$
$Q \leftarrow \langle\rangle$ ▷ FIFO queue
while $T \neq \emptyset \vee Q \neq \emptyset$ **do**
  if $Q \neq \emptyset$ **then**
    $current \leftarrow Q[1]$
    $Q \leftarrow Q[2:]$
  else
    $current \leftarrow T[1]$
    $T \leftarrow T[2:]$
  end if
  $O \leftarrow O$ ++ $\langle current \rangle$
  for all $n \in N[current]$ **do**
    if $n \notin O \wedge n \notin Q$ **then**
      $Q \leftarrow Q$ ++ $\langle n \rangle$
      $T \leftarrow [\, t \in T : t \neq n \,]$
    end if
  end for
end while
return O

neighbours are then added to a first-in-first-out (FIFO) queue. As long as the neighbour queue is not empty, tiles are popped off the front of the queue and their respective neighbours are added to the end. Whenever a tile is added to the neighbour queue, it is removed from the input list. If the neighbour queue is empty, the next element is taken off the input queue. The algorithm ends when the input list and the neighbour queue are both empty. This facilitates efficient processing not only of rectangular images, but also of arbitrary areas of interest. Tiles without neighbours are added at the end of the queue. To identify a tile’s neighbours, we check each tile’s bounds for overlap with all other tiles’ bound coordinates. In case of strictly rectangular images, it is also possible to compute neighbourhood relationships by comparing row and column indices.

2.2 Local NMS

The inference stage produces and enqueues per-tile metadata, which a postprocessor consumes, as illustrated by Figure 2(b) and Algorithm 2. This enables postprocessing in parallel to inference, reducing the postprocessing overhead. Object detection results are shared between processes via shared memory rather than queues to reduce computational overhead and enable scalability for very dense scenes and more sophisticated object shapes, with only metadata being added to the queue. The postprocessor initializes an empty map that serves as both a data buffer and a repository for processed tiles. When a new detection result arrives from the queue, it is immediately added to the buffer. The postprocessor then iterates over the buffer to verify that all neighbours of each pending tile prediction are either pending and stored in the buffer or already processed. If this condition is met, it aggregates detections from neighbouring tiles in overlapping regions and applies NMS and other necessary postprocessing steps to all detections within the tile’s bounds. Finally, it emits the consolidated detections, updates the buffer state, and marks the tile as processed. Processed detections are removed from the pending neighbours to avoid duplicate work. The preordering of the tiles described by Algorithm 1 as well as the continuous checking of the pending tiles ensure that the buffer remains small enough to evaluate at each step. Image metadata is passed through the queue to facilitate the correct mapping from tiles to their corresponding images so that multiple images can be processed consecutively without interruption.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.

Figure 4. Example scene from DOTA-v2.0 dataset with HBB. (Ding et al., 2021a).

Algorithm 2 Pseudocode for Local NMS

Require: Q: Producer queue ▷ Populated by inference process
pending ← ⟨⟩ ▷ Buffer
processed ← ⟨⟩
while $Q \neq \emptyset$ **do**
  tile ← Q.get()
  pending ← *pending* ++ ⟨*tile*⟩
  for all $t \in$ *pending* **do**
    $N \leftarrow t$.neighbours ▷ Neighbour tiles
    if $N \subseteq$ (*processed* ∪ *pending*) **then**
      for all $n \in N$ **do**
        overlap ← { *bbox* | *bbox* $\in n$.bboxes | *bbox* $\cap\, t$.bounds > 0 }
        t.bboxes ← t.bboxes ++ n.bboxes[*overlap*]
        n.bboxes ← n.bboxes[¬*overlap*]
      end for
      processed ← *processed* ++ nms(t.bboxes)
      pending ← [$p \in$ *pending* : $p \neq t$]
    end if
  end for
end while

Here, t.neighbours returns the list of tiles adjacent to t; t.bboxes returns the list of detected bounding boxes on t; and $\mathrm{nms}(B)$ applies greedy non-maximum suppression to the list of bounding boxes B.

3. Experimental Setup

To assess our method, we use three transformer-based object detection models, one for HBB and two for OBB, and two remote sensing benchmark datasets that include very large and dense scenes, introduced in section 3.1. Inference model characteristics and configurations are described in Section 3.2. We compare runtime as well as qualitative results between local NMS and a standard global NMS baseline.

3.1 Datasets

Experiments are conducted on the open remote sensing object detection benchmark datasets DOTA-v2.0 (Ding et al., 2021a) and Izembek Lagoon Birds (Weiser et al., 2022). DOTA-v2.0 consists of aerial and satellite images sized between 800×800 and over 20,000×20,000 pixels. The validation set contains 593 images with 81,048 instances of 18 classes, including traffic and sports infrastructure as well as vehicles, planes, and helicopters. Object sizes vary widely, with average HBB side lengths between 3 and 3120 pixels and an average side length of 34 pixels. Both HBB and OBB are available for each annotated object. An example for a large dense scene with HBB annotations is shown in Figure 4. Figure 5 is a different example from the same dataset with OBB. The official validation split is used for testing as labels for the test split are not publicly available.

Dataset	Size Category	Size [pixels]	Object Density per 1024×1024 pixels	#Images	#Instances
Izembek Lagoon Birds	Large	50,320,896	0-4 5-50 50-58	50 50 1	2667 26,444 2800
DOTA-v2.0	Small	259,350-4,999,999	0-4 5-50 50-708	125 176 102	322 5358 20,774
	Intermediate	5,000,000-19,999,999	0-4 5-50 50-708	93 43 17	1642 7224 17227
	Large	20,000,000-806,504,000	0-4 5-50 50-708	17 15 5	1739 9070 17,692
	All		0-4 5-50 50-708	235 234 124	3703 21,652 55,693

Table 1. Number of test images and object instances per size and density category.

Additionally, we assess the workflow using the Izembek Lagoon Birds dataset (Weiser et al., 2022) (Figure 6), a benchmark dataset from the field of wildlife monitoring with aerial images of waterfowl which have been divided into five bird categories. The randomly chosen test dataset contains 697 images sized 8688×5792 pixels with 77,700 instances annotated with HBB. Each bounding box measures 28×28 pixels.

We divide each of the test sets by image size (small, intermediate, large) and object density (low, high, very high), as listed in Table 1. Object density is defined as the average number of objects per tile across the image, with “low” less than 5 and “very high” greater or equal to 50, a tile being 1024×1024 pixels. All Izembek Lagoon Bird images fall into the large size category. The DOTA-v2.0 scene with the highest density contains over 700 objects per tile, whereas maximum object density in the Izembek Lagoon Birds dataset is limited to 58 objects per tile. In both datasets, low to moderate object densities are more common than very high density values. In the Izembek Lagoon Birds dataset, only one test image contains more than 50 objects per tile. Due to the large size of the images and the test set, we randomly select a subset of 50 images for each of the other categories.

3.2 Implementation Details

In the following section, the main settings for model training and inference used for performance benchmarking are described. Furthermore, we mention any relevant details of the

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.

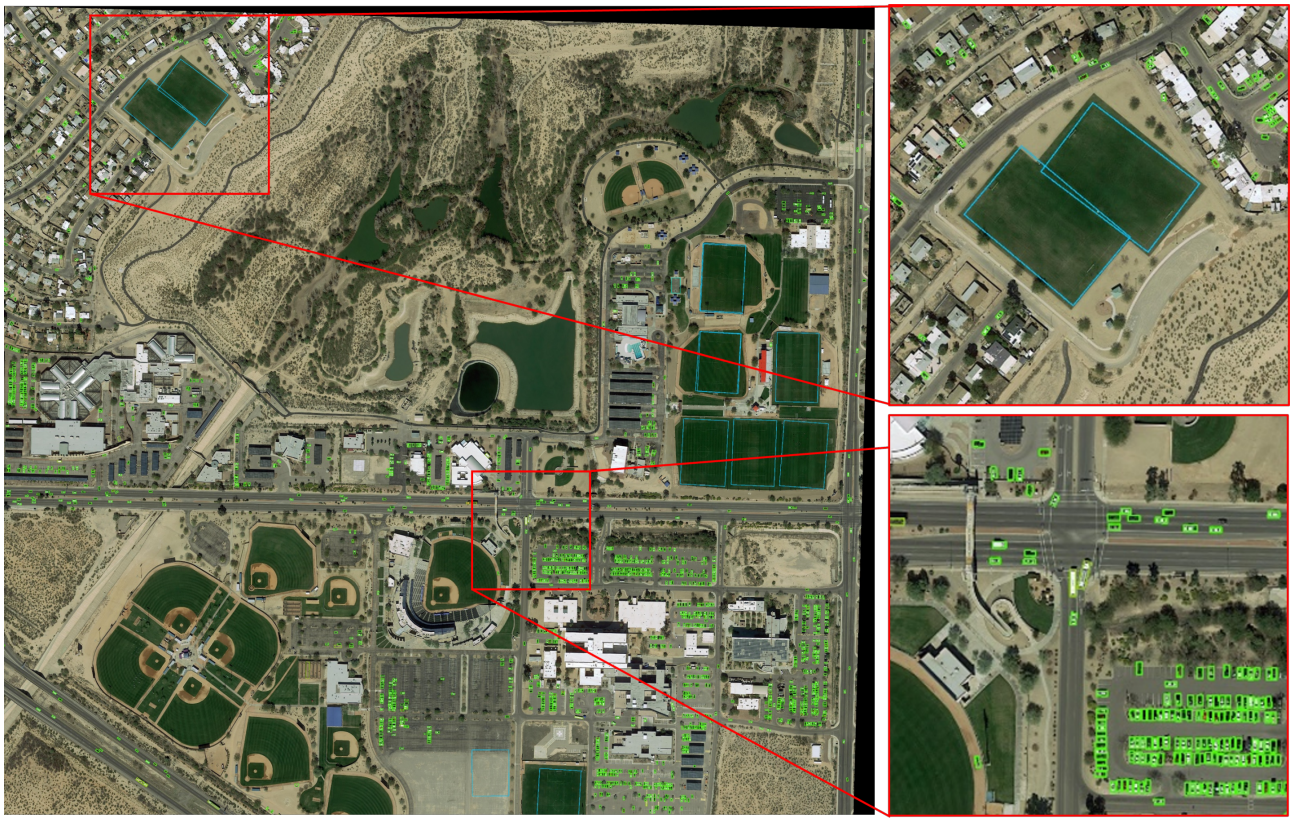


Figure 5. Example scene from DOTA-v2.0 dataset with OBB. (Ding et al., 2021a).

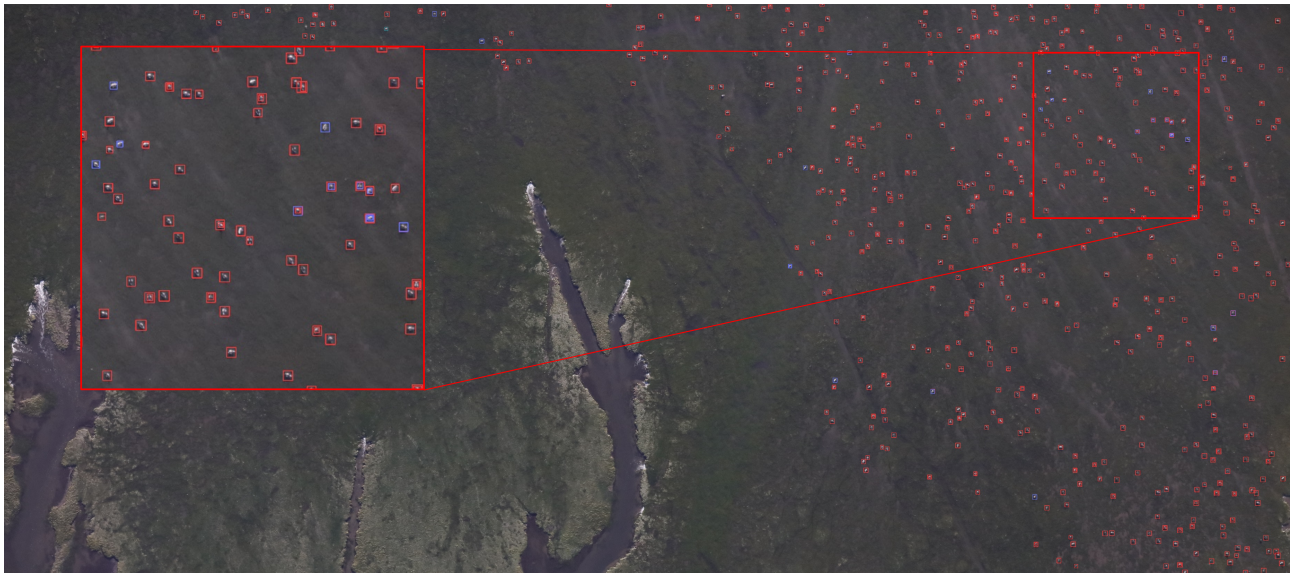


Figure 6. Example scene from the Izembek Lagoon Birds dataset (Weiser et al., 2022).

prototype implementation of local NMS that go beyond the algorithm descriptions in Section 2.

We use a state-of-the-art DINO (5-scale) model with a Swin-L backbone (Zhang et al., 2022), trained in MMDetection (Chen et al., 2019) for HBB detections. The model is trained on DOTA-v2.0 HBB as well as the Izembek Lagoon Birds dataset using AdamW optimizer (Loshchilov et al., 2017) with a learning rate of 1e-4. For DOTA-v2.0, the official training and validation split is used. The training set consists of 1830 images with 268,627 instances. The Izembek Lagoon dataset is split randomly into a training set with 4110 images containing 428,557 instances and a validation set with 175 images and 14,475 instances, as well as the test set described in Section 3.1. The model is trained for 24 epochs on DOTA-v2.0 and 36 epochs on the Izembek Lagoon dataset, respectively.

MessDet, a single-stage oriented object detection architecture (Wu et al., 2025), is trained on the DOTA-v2.0 training set with OBB for 36 epochs with AdamW optimizer (Loshchilov et al., 2017) with an initial learning rate of 1.25e-4 and a weight decay of 0.05, with pretrained COCO-weights.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.

Dataset	Size Category	Object Density per 1024x1024 pixels	DINO (HBB)					OrientedFormer (OBB)					MessDet (OBB)				
			NMS		Local NMS		runtime reduction	NMS		Local NMS		runtime reduction	NMS		Local NMS		runtime reduction
			[s]				[%]	[s]				[%]	[s]				[%]
			mean	σ	mean	σ		mean	σ	mean	σ		mean	σ	mean	σ	
Izembek Lagoon Birds	Large	0-4	33.9	0.11	**33.35**	0.11	1.62	-	-	-	-	-	-	-	-	-	-
		5-49	34.17	0.10	**33.49**	0.33	1.99	-	-	-	-	-	-	-	-	-	-
		50-58	34.70	0.22	**34.42**	0.26	0.81	-	-	-	-	-	-	-	-	-	-
DOTA-v2.0	Small	0-4	1.10	0.03	**1.09**	0.03	0.91	**0.38**	0.00	**0.38**	0.01	0.00	**0.50**	0.01	**0.50**	0.01	0.00
		5-49	1.33	0.03	**1.31**	0.05	1.50	0.48	0.01	**0.45**	0.01	6.25	0.65	0.01	**0.61**	0.01	6.15
		50-708	1.62	0.04	**1.44**	0.03	11.11	0.80	0.02	**0.52**	0.01	35.00	1.10	0.03	**0.74**	0.01	32.73
	Intermediate	0-4	10.49	0.17	**10.37**	0.24	1.14	3.65	0.04	**3.45**	0.03	5.48	4.76	0.04	**4.57**	0.02	3.99
		5-49	8.14	0.24	**7.88**	0.12	3.19	3.01	0.05	**2.73**	0.04	9.30	3.87	0.04	**3.60**	0.03	6.98
		50-708	7.95	0.24	**7.20**	0.21	9.43	4.78	0.17	**2.61**	0.05	45.40	5.65	0.17	**3.56**	0.07	36.99
	Large	0-4	52.48	2.53	**51.75**	2.22	1.39	18.70	0.18	**18.47**	0.10	1.23	24.15	0.16	**23.79**	0.27	1.49
		5-49	23.97	0.77	**23.16**	1.06	3.38	9.51	0.22	**8.03**	0.10	15.56	12.24	0.17	**10.65**	0.14	12.99
		50-708	23.42	0.35	**21.34**	0.59	8.88	15.79	0.68	**7.67**	0.13	51.42	32.40	1.69	**12.71**	0.29	60.77
	All	0-4	8.46	0.07	**8.37**	0.13	1.06	3.00	0.03	**2.87**	0.04	4.33	3.90	0.03	**3.77**	0.02	3.33
		5-49	4.00	0.03	**3.93**	0.09	1.75	1.51	0.01	**1.38**	0.12	8.61	1.98	0.02	**1.80**	0.02	9.09
		50-708	3.36	0.04	**3.02**	0.03	10.12	1.90	0.03	**1.09**	0.02	42.63	2.93	0.09	**1.51**	0.02	48.46

Table 2. Processing time per size and density category, mean and standard deviation in seconds per image over 10 runs. The lower mean processing time for each model and data category is highlighted in bold type.

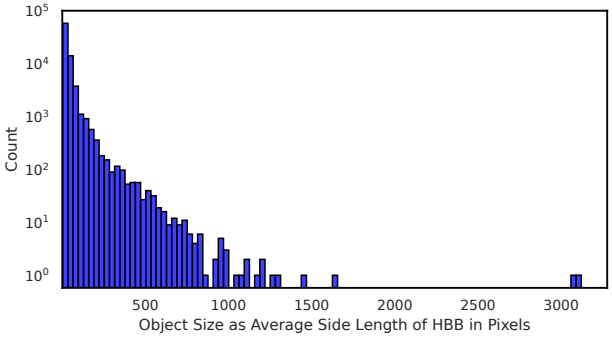


Figure 7. Object sizes as average side length of HBB in DOTA-v2.0.

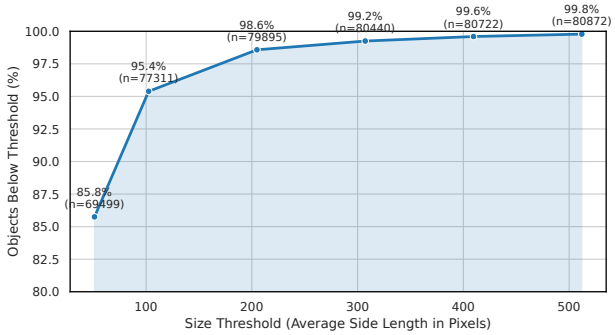


Figure 8. Percentage of objects below a size in pixels as average side length of the HBB in DOTA-v2.0. Labeled points correspond to 5%, 10%, 20%, 30%, 40%, and 50% of the tile size.

OrientedFormer, an end-to-end transformer-based oriented object detector with a Resnet50 backbone (Zhao et al., 2024), is also applied to OBB detection on DOTA-v2.0. The configuration and pretrained weights are available in the AI4RS framework (Ding, 2025).

The preordering algorithm and producer-consumer pattern used for local NMS do not require any particular training configuration and can be combined with any pre-trained object detection model that produces either HBB or OBB and any framework for tile-wise inference. We use the Python-based framework SAHI (Akyon et al., 2022) as a baseline and integrate our algorithms with the existing tiling and inference workflows. Comparison algorithms for the identification of overlaps between tiles are implemented efficiently with Numpy (Harris et al., 2020) and Cython. Parallelization is achieved through Python's Multiprocessing library. For NMS, we use the greedy NMS algorithm included in the SAHI framework (Akyon et al., 2022) for horizontal bounding boxes and MMCV's quadri NMS (MMCV Contributors, 2018) for OBB, with a match threshold over the IoU of 0.5, respectively. Quadri NMS efficiently computes greedy NMS over OBB with four corner coordinates in the format $\{x_0, y_0, x_1, y_1, x_2, y_2, x_3, y_3\}$.

The key hyperparameters for tile-based inference are tile size and tile overlap. Tile size is determined by model characteristics and available GPU memory. Larger tiles reduce edge effects and the absolute overlap area. We use a fixed tile size of 1024×1024 across all experiments to match the trained models and ensure comparability. The number of tiles per scene grows rapidly as the overlap approaches the tile size (approximately with the inverse square of the stride). Large overlaps produce many redundant predictions, whereas overlaps smaller than the maximum expected object size cause incomplete predictions at tile edges. Because the largest objects in DOTA-v2.0 exceed the tile size, we consider the distribution of object sizes (Figure 7). Given the wide range of bounding-box sizes, we consider overlaps of 5%, 10%, 20%, 30%, 40%, and 50% of the tile size. Figure 8 shows the percentage of DOTA-v2.0 objects whose size is below each overlap threshold. We select a 20% overlap as the best tradeoff between coverage and a moderate overlap relative to tile size. The same hyperparameters are applied to the Izembek Lagoon Birds dataset for consistency.

3.3 Metrics

A Tesla V100-PCIE GPU with 32 GB is used for inference. We measure the combined runtime for image tiling and reordering, inference, and postprocessing in seconds without taking model or task initialization into account. Processing time is averaged per category over ten inference and postprocessing runs per model and image. We also compute the average per-image standard deviation over all images and runs per category. The runtime reduction metric is computed as follows:

$$\text{Runtime reduction } (\%) = (1 - \frac{T_{\text{Local NMS}}}{T_{\text{NMS}}}) \times 100 \quad (1)$$

To ensure that the quality of results remains stable, we compare the mean average precision (mAP), between our approach and the baseline, with IoU thresholds of 0.5 and 0.5 to 0.95.

4. Results and Discussion

In this section, we describe and discuss the results of the experiments for runtime comparison between local NMS and standard

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.

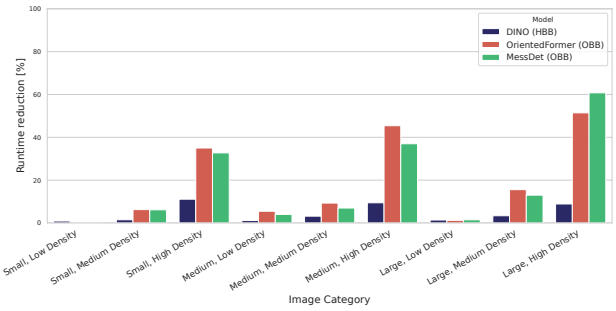


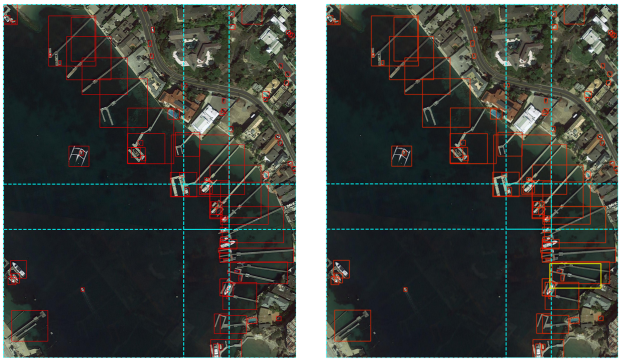
Figure 9. Runtime reduction in % per model type by image size and density categories with 20% tile overlap.

sequential postprocessing in a tile-wise object detection inference workflow. First, we show quantitative results for each of the models and datasets. Then, we qualitatively evaluate the results of local NMS. Finally, we briefly discuss the implications of our findings.

4.1 Quantitative Results

The results of the runtime measurements described in Section 3 with a tile overlap of 20% are presented in Table 2. For both datasets, the runtime of local NMS is shorter across all categories for all three models, except for both OBB detectors and small images with few objects, where no change was observed. For the DINO model with HBB and DOTA-v2.0, the runtime reduction is in the range of 0.91% to 11.11% and for the OrientedFormer model with OBB and quadri NMS, we observe a runtime reduction of 0% to 51.42%; similarly, the runtime reduction for MessDet ranges between 0% and 60.77%. In all three models, the runtime reduction is greater for dense images. However, its dependency on image size differs slightly between models. For OrientedFormer and MessDet, the runtime reduction increases with image size. This is also the case for DINO, except with very dense scenes, where runtime reduction decreases as image size increases. The runtime reduction per category for each of the three models is visualized by Figure 9. Observed standard deviations are consistently in the range of approximately 1-5% of the respective mean processing time, with no pronounced differences between models or between standard and local NMS. Absolute mean processing times are highly dependent on image size, ranging from less than 0.5 seconds to over 50 seconds, (Table 2). OrientedFormer and MessDet exhibit shorter processing times than DINO across all categories. The maximum relative runtime reduction across both models and all categories was observed for MessDet on large, high-density images, for which the mean processing time was reduced by 60.77%, followed by OrientedFormer with 51.42% runtime reduction in the same category. The mean runtime reduction across all object density categories for all image sizes is 4.31% for the DINO model, 18.52% for OrientedFormer, and 20.30% for MessDet. The mAP remains stable with differences between standard and local NMS not exceeding 0.2 percentage points at IoU thresholds of 0.5 and 0.5 to 0.95.

Table 2 and Figure 9 show that even though local NMS does not provide significant runtime reduction in cases with few objects, runtime remains robust and the additional processing steps carried out by Algorithm 1 and Algorithm 2, i.e. tile reordering and the collecting of bounding boxes from overlapping neighbour tile areas, rarely slow the processing down below the speed of standard global NMS. Table 2 shows differences of per-image runtime distributions for the DOTA-v2.0 dataset across



(a) Example with standard NMS. (b) Example with local NMS.

Figure 10. Example DOTA-v2.0 scene illustrating equivalent results of local and standard global NMS, with the exception of an additional bounding box in local NMS (yellow) that is processed with one tile while an overlapping bounding box with higher confidence score remains in another tile. Dashed lines symbolize tile borders.

the three benchmark models. While OrientedFormer is fastest overall, the most marked runtime reduction is observed for the MessDet model.

In the case of the Izembek Lagoon Birds dataset, only minor runtime reductions between 0.81% and 1.99% were observed with a tile overlap of 20%, with no change to the mAP. The runtime reduction does not appear to be correlated to object density. As noted in Section 3.1, the ”dense” category contains only one image. Local NMS occasionally retains slightly more bounding boxes than standard global NMS; across five large dense scenes from the DOTA-v2.0 dataset, OrientedFormer with local NMS yields 10,325 object detections vs. 10,321 object detections retained by standard global NMS.

Different tile overlaps influence both absolute processing times and runtime reductions. The results of experiments with overlaps of 5%, 10%, 30%, 40%, and 50% are listed in Appendix A, Table 3. Small overlaps lead to lower overall runtime and lower runtime reduction. In some cases, an increase in runtime is observed, the most significant being 3.97% for OrientedFormer and large images with low object density processed with a tile overlap of 5%. However, the mean runtime reduction across all image categories and models is positive even with a 5% tile overlap, with 0.55% for DINO on the Izembek Lagoon Birds dataset and 5.30% on DOTA-v2.0 and 15.47% and 16.94% for OrientedFormer and MessDet on DOTA-v2.0, respectively. The overall highest runtime reduction is measured at 67.23% for MessDet at 50% tile overlap with large dense images. Different overlaps have no significant impact on mAP.

4.2 Qualitative Results

Qualitative, sample-based visual analysis of the results suggests that local NMS yields equivalent results to standard global NMS, with very few exceptions in cases where a bounding box that has already been processed in an earlier step and thus removed from a tile has high overlap and a higher confidence score than a remaining bounding box that is processed in a later iteration, as illustrated by Figure 10.

4.3 Discussion

Results, presented in Table 2 and Figure 9 as well as Appendix A, indicate that local NMS significantly and consist-

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.

ently reduces runtime compared to the baseline for three state-of-the-art transformer-based object detection models over remote sensing images in a tiling-based inference workflow. The runtime reduction is especially significant for dense large images and OBB. However, runtime reduction of DINO (HBB) on the Izembek Lagoon Birds dataset is in the same range as values measured with the same model on the DOTA-v2.0 test sets with low or intermediate density, with a maximum runtime reduction of 1.99%. This might be due to relatively lower object densities in the Izembek Lagoon Birds dataset as well as very unequal density distributions within individual images, where flocks of birds only cover part of a very large area, as illustrated by Figure 6.

Even though relative runtime reduction slightly decreases with increasing image sizes for the DINO (HBB) model, our method scales well to large images and high object densities. Its runtime rarely exceeds the runtime of standard global NMS, which indicates that our method is robust to real-world scenarios with variable inputs and provides a significant increase in efficiency, especially for large dense scenes as used e.g. in traffic or wildlife monitoring, where arbitrarily oriented small objects such as vehicles need to be identified in near-real time.

We observe that runtime reduction is particularly significant for OBB detectors, as illustrated by Figure 9. This is due to the relative runtime of a more sophisticated postprocessing algorithm being high compared to very efficient model architectures used for the runtime benchmarking experiments. The greatest improvement can be expected by pipelining if both processes take approximately the same amount of time. If postprocessing takes very little time compared to inference, as is the case for the relatively heavy-weight DINO model with highly efficient HBB NMS, only small to moderate improvements can be achieved.

Tile size and tile overlap have a significant impact on the processing time. A larger tile size decreases the amount of overlapping areas and thus reduces overall runtime and the number of redundant objects, which in turn decreases the potential for runtime reduction through local NMS. Tile overlap should be greater than the largest expected object size. However, a larger overlap increases redundancy and consequently runtime.

As a rule, local NMS yields equivalent results to global NMS. However, we observed that in rare cases, slightly more bounding boxes are returned by our workflow compared to the baseline. This is due to local NMS only taking bounding boxes into account that overlap with the current tile at each iteration and removing processed bounding boxes from pending tiles. This issue can be mitigated by keeping track of processed bounding boxes and including them in further iterations, by running a final consolidation step over all remaining bounding boxes after all tiles have been received, or by checking bounding boxes for overlap with each other rather than the tile. However, these approaches introduce additional computational and memory overhead at the postprocessing stage. Given that the observed differences are small and confined to specific cases, the associated quality trade-off is not expected to be significant relative to the achieved efficiency gains.

Local NMS has the potential to markedly improve usability for large dense scenes in interactive applications. Partial results can be consumed and visualized while inference is still in progress, offering near real-time feedback on detection quality and object counts. Additionally, local NMS reduces peaks in memory use by releasing processed boxes. This implies reduced hardware requirements compared to standard global NMS especially on large dense scenes. Further experiments are necessary to confirm the usability of the method on edge devices. Multi-GPU systems can be used to distribute tiles during inference and combine them in a final consolidation step. Since local NMS does not depend on tiles being processed in a certain order, we expect that it would integrate well in such systems.

5. Conclusion

We present a simple and effective approach to parallelizing object detection inference and postprocessing in tiling-based inference workflows. Experimental results show that the method significantly decreases runtime and scales well to very large, dense scenes. The improvement is particularly significant for OBB detectors which require sophisticated postprocessing algorithms such as quadri NMS. The runtime can be reduced by up to 60.77% for this use case.

For our further research, we expect that our method scales well when applied to instance segmentation approaches and can be extended to include further complex postprocessing algorithms such as non-maximum merging or polygon-based postprocessing. Future research will include additional benchmarking on different hardware configurations. We expect the method to be well suited to near-real time inference workflows on edge devices and multi-GPU systems. Efficiency and simplicity of the method might further increase with integration into the newest Python version, Python 3.14, which, unlike previous versions, allows true parallel multi-threading, thus eliminating the need for multiple parallel processes (Python Software Foundation, 2025). Integration into other, more efficient programming languages used for object detection inference is also possible.

References

Akyon, F. C., Altinuc, S. O., Temizel, A., 2022. Slicing aided hyper inference and fine-tuning for small object detection. *2022 IEEE international conference on image processing (ICIP)*, IEEE, 966–970.

Bodla, N., Singh, B., Chellappa, R., Davis, L. S., 2017. Soft-nms–improving object detection with one line of code. *Proceedings of the IEEE international conference on computer vision*, 5561–5569.

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S., 2020. End-to-end object detection with transformers. *European conference on computer vision*, Springer, 213–229.

Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C. C., Lin, D., 2019. MMDetection: Open MMLab Detection Toolbox and Benchmark. *arXiv preprint arXiv:1906.07155*.

Cohen, T. S., Welling, M., 2016. Group equivariant convolutional networks.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., Fei-Fei, L., 2009. Imagenet: A large-scale hierarchical image database. *2009 IEEE conference on computer vision and pattern recognition*, Ieee, 248–255.

Ding, J., Xue, N., Xia, G.-S., Bai, X., Yang, W., Yang, M., Belongie, S., Luo, J., Datcu, M., Pelillo, M., Zhang, L., 2021a. Object Detection in Aerial Images: A Large-Scale Benchmark and Challenges. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1-1.

Ding, J., Xue, N., Xia, G.-S., Bai, X., Yang, W., Yang, M. Y., Belongie, S., Luo, J., Datcu, M., Pelillo, M. et al., 2021b. Object detection in aerial images: A large-scale benchmark and challenges. *IEEE transactions on pattern analysis and machine intelligence*, 44(11), 7778–7796.

Ding, Z., 2025. Ai4rs. https://github.com/wokaikaixinxin/ai4rs. Accessed 2025-11-06.

Girshick, R., Donahue, J., Darrell, T., Malik, J., 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 580–587.

Guo, C., Zhang, B., 2023. FS-DINO: improved detection method for full-scale objects based on DINO from high-resolution remote sensing imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 16, 10381–10393.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., Oliphant, T. E., 2020. Array programming with NumPy. *Nature*, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2.

Hua, W., Chen, Q., 2025. A survey of small object detection based on deep learning in aerial images. *Artificial Intelligence Review*, 58(6), 1–67.

Li, Z., He, G., Fu, H., Chen, Q., Shangguan, B., Feng, P., Jin, S., 2023. Rs dino: A novel panoptic segmentation algorithm for high resolution remote sensing images. *2023 11th International Conference on Agro-Geoinformatics (Agro-Geoinformatics)*, IEEE, 1–5.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., Belongie, S., 2017. Feature pyramid networks for object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2117–2125.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C. L., 2014. Microsoft coco: Common objects in context. *European conference on computer vision*, Springer, 740–755.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A. C., 2016. Ssd: Single shot multibox detector. *European conference on computer vision*, Springer, 21–37.

Liu, Z., Chen, C., Huang, Z., Chang, Y. C., Liu, L., Pei, Q., 2024. A Low-Cost and Lightweight Real-Time Object-Detection Method Based on UAV Remote Sensing in Transportation Systems. *Remote Sensing*, 16(19). https://www.mdpi.com/2072-4292/16/19/3712.

Loshchilov, I., Hutter, F. et al., 2017. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5(5), 5.

Michel, A., 2025. Deep Learning for Environmental Remote Sensing Image Understanding: Analyzing Dust and Anthropogenic Objects Across Varying Scales and Densities. PhD thesis, Karlsruher Institut für Technologie (KIT).

Michel, A., Gross, W., Hinz, S., Middelmann, W., 2022. Armnms: Shape based non-maximum suppression for instance segmentation in large scale imagery. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2, 291–298.

Michel, A., Mispelhorn, J., Schenkel, F., Gross, W., Middelmann, W., 2020. An approach to improve detection in scenes with varying object densities in remote sensing. *Image and Signal Processing for Remote Sensing XXVI*, 11533, SPIE, 107–113.

MMCV Contributors, 2018. MMCV: OpenMMLab computer vision foundation. https://github.com/open-mmlab/mmcv.

Python Software Foundation, 2025. What’s new in python 3.14. https://docs.python.org/3/whatsnew/3.14.html. Accessed 2025-10-30.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 779–788.

Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28.

Sun, X., Wang, P., Yan, Z., Xu, F., Wang, R., Diao, W., Chen, J., Li, J., Feng, Y., Xu, T. et al., 2022. FAIR1M: A benchmark dataset for fine-grained object recognition in high-resolution remote sensing imagery. *ISPRS Journal of Photogrammetry and Remote Sensing*, 184, 116–130.

Tong, H., Yuan, J., Zhang, J., Wang, H., Li, T., 2024. Real-Time Wildfire Monitoring Using Low-Altitude Remote Sensing Imagery. *Remote Sensing*, 16(15). https://www.mdpi.com/2072-4292/16/15/2827.

Weiser, E. L., Flint, P. L., Marks, D. K., Shults, B. S., Wilson, H. M., Thompson, S. J., Fischer, J. B., 2022. Aerial photo imagery from fall waterfowl surveys, izembek lagoon, alaska, 2017-2019. *US Geological Survey (USGS) Data Release*, 637.

Wu, X., Wang, X., Zhu, X., Yang, L., Liu, J., Hu, X., 2025. Measuring the impact of rotation equivariance on aerial object detection. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 7329–7339.

Xia, G.-S., Bai, X., Ding, J., Zhu, Z., Belongie, S., Luo, J., Datcu, M., Pelillo, M., Zhang, L., 2018. Dota: A large-scale dataset for object detection in aerial images. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, H., Li, F., Liu, S., Zhang, L., Su, H., Zhu, J., Ni, L. M., Shum, H.-Y., 2022. Dino: Detr with improved denoising anchor boxes for end-to-end object detection.

Zhao, J., Ding, Z., Zhou, Y., Zhu, H., Du, W.-L., Yao, R., El Saddik, A., 2024. OrientedFormer: An End-to-End Transformer-Based Oriented Object Detector in Remote Sensing Images. *IEEE Transactions on Geoscience and Remote Sensing*, 62, 1-16.

Appendix A. Experiments with Different Tile Overlap Settings

Table 3 shows results for different tile overlaps, expressed as percentages of the fixed tile size of 1024×1024 pixels. These results supplement and are generally consistent with Table 2. Local NMS occasionally exhibits longer runtimes than the baseline in sparse scenes, particularly at smaller overlaps. The greatest runtime reductions occurred in dense scenes with high overlaps. OrientedFormer and MessDet exhibit larger runtime reductions than DINO, particularly in dense scenes, with up to 67% for MessDet at 50% overlap on large DOTA-v2.0 images. This indicates that local NMS reduces postprocessing overhead caused by many redundant predictions and OBB postprocessing, but also introduces some overhead through tile reordering and bounding box consolidation.

Tile Overlap	Dataset	Size Category	Object Density per 1024x1024 pixels	DINO (HBB)					OrientedFormer (OBB)					MessDet (OBB)				
				NMS		Local NMS		runtime reduction	NMS		Local NMS		runtime reduction	NMS		Local NMS		runtime reduction
				[s]				[%]	[s]				[%]	[s]				[%]
				mean	σ	mean	σ		mean	σ	mean	σ		mean	σ	mean	σ	
5%	Izembek Lagoon Birds	Large	0-4	24.41	0.21	**24.11**	0.16	1.23	-	-	-	-	-	-	-	-	-	-
			5-49	24.57	0.12	**24.25**	0.09	1.30	-	-	-	-	-	-	-	-	-	-
			50-58	**24.93**	0.20	25.15	0.23	-0.88	-	-	-	-	-	-	-	-	-	-
	DOTA-v2.0	Small	0-4	**0.99**	0.00	1.00	0.01	-1.01	**0.35**	0.00	**0.35**	0.01	0.00	**0.46**	0.01	**0.46**	0.01	0.00
			5-49	1.24	0.01	**1.22**	0.02	1.61	0.46	0.01	**0.42**	0.01	8.70	0.62	0.01	**0.58**	0.01	6.45
			50-708	1.50	0.01	**1.35**	0.02	10.00	0.74	0.01	**0.49**	0.01	33.78	0.99	0.01	**0.70**	0.01	29.29
		Intermediate	0-4	10.02	0.05	**9.89**	0.05	1.30	3.55	0.05	**3.39**	0.03	4.51	4.68	0.04	**4.53**	0.06	3.21
			5-49	7.11	0.06	**6.97**	0.08	1.97	2.70	0.04	**2.47**	0.03	8.52	3.52	0.06	**3.26**	0.04	7.39
			50-708	7.19	0.06	**6.57**	0.10	8.62	4.24	0.10	**2.41**	0.03	43.16	4.93	0.11	**3.33**	0.10	32.45
		Large	0-4	**39.21**	0.20	39.49	0.26	-0.71	**14.59**	0.12	15.16	0.15	-3.91	**19.15**	0.63	19.56	0.69	-2.14
			5-49	18.57	0.11	**18.03**	0.08	2.91	7.58	0.10	**6.62**	0.07	12.66	9.79	0.14	**8.77**	0.18	10.42
			50-708	18.64	0.19	**17.09**	0.21	8.32	11.73	0.29	**6.33**	0.13	46.04	20.74	0.27	**9.28**	0.28	55.26
		All	0-4	7.37	0.07	**7.35**	0.08	0.27	2.65	0.03	**2.59**	0.05	2.26	3.48	0.02	**3.41**	0.05	2.01
			5-49	3.63	0.64	**3.38**	0.07	6.89	1.32	0.01	**1.27**	0.17	3.79	1.77	0.09	**1.62**	0.06	8.47
			50-708	2.97	0.02	**2.71**	0.03	8.75	1.66	0.03	**0.99**	0.02	40.36	2.33	0.07	**1.39**	0.03	40.34
10%	Izembek Lagoon Birds	Large	0-4	31.39	0.12	**30.92**	0.17	1.50	-	-	-	-	-	-	-	-	-	-
			5-49	31.63	0.21	**30.96**	0.18	2.12	-	-	-	-	-	-	-	-	-	-
			50-58	32.70	0.68	**32.14**	0.24	1.71	-	-	-	-	-	-	-	-	-	-
	DOTA-v2.0	Small	0-4	**1.02**	0.01	1.03	0.01	-0.98	**0.36**	0.00	**0.36**	0.01	0.00	0.48	0.01	**0.47**	0.01	2.08
			5-49	1.26	0.01	**1.24**	0.01	1.59	0.46	0.00	**0.43**	0.00	6.52	0.63	0.00	**0.60**	0.01	4.76
			50-708	1.53	0.02	**1.36**	0.02	11.11	0.75	0.01	**0.49**	0.01	34.67	1.01	0.01	**0.71**	0.01	29.70
		Intermediate	0-4	10.10	0.05	**10.01**	0.06	0.89	3.53	0.02	**3.39**	0.03	3.97	4.69	0.04	**4.54**	0.04	3.20
			5-49	7.28	0.05	**7.10**	0.05	2.47	2.74	0.04	**2.50**	0.04	8.76	3.59	0.10	**3.34**	0.03	6.96
			50-708	7.39	0.08	**6.67**	0.04	9.74	4.42	0.08	**2.45**	0.03	44.57	5.08	0.12	**3.35**	0.05	34.06
		Large	0-4	**42.92**	0.20	43.20	0.16	-0.65	**15.77**	0.26	16.07	0.13	-1.90	**20.54**	0.25	20.81	0.23	-1.31
			5-49	20.84	0.22	**20.13**	0.13	3.41	8.40	0.08	**7.22**	0.07	14.05	10.80	0.14	**9.63**	0.12	10.83
			50-708	19.68	0.32	**17.92**	0.14	8.94	12.76	0.26	**6.51**	0.13	48.98	22.02	0.31	**9.44**	0.30	57.13
		All	0-4	7.65	0.03	**7.62**	0.03	0.39	2.74	0.02	**2.64**	0.03	3.65	3.62	0.05	**3.53**	0.03	2.49
			5-49	3.63	0.02	**3.53**	0.01	2.75	1.39	0.01	**1.25**	0.02	10.07	1.83	0.01	**1.68**	0.03	8.20
			50-708	3.06	0.03	**2.74**	0.01	10.46	1.74	0.03	**1.01**	0.01	41.95	2.40	0.04	**1.41**	0.02	41.25
30%	Izembek Lagoon Birds	Large	0-4	42.83	0.16	**41.91**	0.16	2.15	-	-	-	-	-	-	-	-	-	-
			5-49	43.13	0.15	**42.08**	0.09	2.43	-	-	-	-	-	-	-	-	-	-
			50-58	43.94	0.53	**43.48**	0.24	1.05	-	-	-	-	-	-	-	-	-	-
	DOTA-v2.0	Small	0-4	**1.17**	0.01	1.18	0.01	-0.85	0.41	0.00	**0.40**	0.01	2.44	0.54	0.01	**0.53**	0.00	1.85
			5-49	1.40	0.01	**1.37**	0.02	2.14	0.51	0.00	**0.47**	0.00	7.84	0.70	0.00	**0.65**	0.01	7.14
			50-708	1.66	0.02	**1.48**	0.01	10.84	0.84	0.02	**0.53**	0.01	36.90	1.12	0.01	**0.78**	0.01	30.36
		Intermediate	0-4	14.28	0.04	**14.03**	0.06	1.75	4.91	0.04	**4.63**	0.03	5.70	6.57	0.06	**6.28**	0.08	4.41
			5-49	9.92	0.06	**9.63**	0.03	2.92	3.65	0.04	**3.32**	0.04	9.04	4.79	0.03	**4.42**	0.05	7.72
			50-708	9.78	0.10	**8.84**	0.08	9.61	5.87	0.13	**3.14**	0.03	46.51	6.61	0.09	**4.47**	0.09	32.38
		Large	0-4	66.86	0.21	**66.34**	0.33	0.78	23.73	0.17	**22.86**	0.14	3.67	31.51	0.32	**30.47**	0.29	3.30
			5-49	30.24	0.23	**29.15**	0.17	3.60	11.87	0.09	**10.04**	0.15	15.42	15.60	0.21	**13.44**	0.18	13.85
			50-708	29.35	0.35	**26.61**	0.26	9.34	19.42	0.21	**9.37**	0.15	51.75	37.70	0.51	**14.14**	0.26	62.49
		All	0-4	11.10	0.03	**10.94**	0.03	1.44	3.87	0.02	**3.67**	0.03	5.17	5.19	0.06	**4.95**	0.03	4.62
			5-49	4.83	0.03	**4.68**	0.02	3.11	1.82	0.01	**1.60**	0.01	12.09	2.40	0.01	**2.19**	0.02	8.75
			50-708	3.90	0.04	**3.50**	0.03	10.26	2.27	0.03	**1.25**	0.02	44.93	3.36	0.06	**1.79**	0.03	46.73
40%	Izembek Lagoon Birds	Large	0-4	56.01	0.23	**54.87**	0.20	2.04	-	-	-	-	-	-	-	-	-	-
			5-49	56.45	0.21	**54.90**	0.29	2.75	-	-	-	-	-	-	-	-	-	-
			50-58	57.26	0.45	**56.17**	0.31	1.90	-	-	-	-	-	-	-	-	-	-
	DOTA-v2.0	Small	0-4	**1.27**	0.01	1.29	0.06	-1.57	0.44	0.00	**0.43**	0.01	2.27	0.58	0.01	**0.57**	0.01	1.72
			5-49	1.54	0.01	**1.50**	0.03	2.60	0.55	0.01	**0.51**	0.01	7.27	0.75	0.01	**0.71**	0.01	5.33
			50-708	1.91	0.02	**1.72**	0.08	9.95	0.93	0.02	**0.61**	0.01	34.41	1.29	0.03	**0.88**	0.02	31.78
		Intermediate	0-4	15.58	0.11	**15.24**	0.14	2.18	5.37	0.05	**5.03**	0.03	6.33	7.10	0.08	**6.70**	0.12	5.63
			5-49	11.53	0.06	**11.29**	0.29	2.08	4.21	0.07	**3.76**	0.03	10.69	5.46	0.07	**5.01**	0.08	8.24
			50-708	11.27	0.10	**10.35**	0.43	8.16	6.94	0.32	**3.62**	0.07	47.84	7.95	0.20	**5.08**	0.13	36.10
		Large	0-4	86.49	0.45	**84.71**	0.53	2.06	30.38	0.25	**28.88**	0.30	4.94	40.23	0.64	**37.88**	0.73	5.84
			5-49	38.68	0.22	**37.00**	0.31	4.34	15.09	0.21	**12.52**	0.16	17.03	19.69	0.26	**16.85**	0.30	14.42
			50-708	37.34	0.47	**33.64**	0.29	9.91	24.74	1.01	**11.72**	0.23	52.63	48.45	2.60	**16.88**	0.53	65.16
		All	0-4	13.14	0.09	**12.86**	0.15	2.13	4.58	0.06	**4.32**	0.06	5.68	6.05	0.11	**5.71**	0.10	5.62
			5-49	5.75	0.04	**5.57**	0.11	3.13	2.16	0.03	**1.93**	0.12	10.65	2.84	0.04	**2.53**	0.04	10.92
			50-708	4.60	0.04	**4.13**	0.06	10.22	2.70	0.09	**1.46**	0.02	45.93	4.11	0.13	**2.06**	0.03	49.88
50%	Izembek Lagoon Birds	Large	0-4	77.51	0.51	**75.51**	0.55	2.58	-	-	-	-	-	-	-	-	-	-
			5-49	77.98	0.51	**75.50**	0.70	3.18	-	-	-	-	-	-	-	-	-	-
			50-58	79.01	0.82	**76.93**	0.97	2.63	-	-	-	-	-	-	-	-	-	-
	DOTA-v2.0	Small	0-4	1.49	0.01	**1.48**	0.02	0.67	0.52	0.01	**0.51**	0.01	1.92	0.69	0.01	**0.68**	0.02	1.45
			5-49	1.69	0.01	**1.64**	0.01	2.96	0.61	0.00	**0.57**	0.01	6.56	0.81	0.01	**0.77**	0.01	4.94
			50-708	2.13	0.02	**1.88**	0.02	11.74	1.05	0.04	**0.69**	0.06	34.29	1.49	0.06	**0.97**	0.02	34.90
		Intermediate	0-4	20.72	0.08	**20.17**	0.05	2.65	7.11	0.10	**6.64**	0.06	6.61	9.40	0.13	**8.87**	0.14	5.64
			5-49	15.00	0.06	**14.46**	0.12	3.60	5.44	0.07	**4.87**	0.07	10.48	7.16	0.16	**6.46**	0.13	9.78
			50-708	14.42	0.13	**12.98**	0.07	9.99	8.57	0.14	**4.55**	0.07	46.91	10.22	0.58	**6.46**	0.19	36.79
		Large	0-4	120.17	0.62	**117.05**	0.89	2.60	41.88	0.33	**39.03**	0.46	6.81	55.55	0.80	**51.80**	0.79	6.75
			5-49	52.29	0.29	**49.68**	0.21	4.99	20.33	0.31	**16.62**	0.16	18.25	26.43	0.66	**22.59**	0.47	14.53
			50-708	50.19	0.47	**45.39**	0.24	9.56	33.97	0.62	**15.51**	0.17	54.34	67.50	5.21	**22.12**	0.39	67.23
		All	0-4	17.81	0.28	**17.36**	0.37	2.53	6.13	0.05	**5.70**	0.06	7.01	8.09	0.07	**7.56**	0.08	6.55
			5-49	7.40	0.08	**7.15**	0.18	3.38	2.76	0.03	**2.38**	0.02	13.77	3.64	0.06	**3.19**	0.04	12.36
			50-708	5.75	0.05	**5.19**	0.11	9.74	3.39	0.02	**1.80**	0.02	46.90	5.35	0.25	**2.54**	0.05	52.52

Table 3. Processing time per tile overlap, size, and density category; mean and standard deviation in seconds per image over 10 runs. The lower mean processing time for each model and data category is highlighted in bold type.

This contribution has been peer-reviewed. The double-blind peer-review was conducted on the basis of the full paper.
https://doi.org/10.5194/isprs-annals-XI-3-2026-81-2026 | © Author(s) 2026. CC BY 4.0 License.