YOShI: YOLO-ODARIS Ship-detection Interface - An Innovative Approach to On-board Data Analysis for Ship-detection

1st Mattia Scaglioni
German Space Operations Center (GSOC)
German Aerospace Center (DLR)
82234, Wessling
scaglioni.mattia@dlr.de

2nd Daniel Herschmann
German Space Operations Center (GSOC)
German Aerospace Center (DLR)
82234, Wessling
daniel.herschmann@dlr.de

3rd Kurt Schwenk
German Space Operations Center (GSOC)
German Aerospace Center (DLR)
82234, Wessling
kurt.schwenk@dlr.de

***Abstract*—The rapid growth of the demand for low-latency information about time-critical events in the field of remote sensing has become a major focus in the aerospace sector. One solution to this demand, is the possibility to extract key information through automated data analysis on-board the satellite. The availability of state-of-the-art AI-based tools has strongly catalysed the development of such systems. Within this context, the *German Aerospace Center* (DLR) develops an "On-board Data Analysis And Real-Time Information System" with ship-detection used as a reference application for AI-based on-board object detection. Despite this large sector, there is no available standard approach for image processing and data handling in this field of application. As a result, the "Yolo-ODARIS Ship-detection Interface" is proposed for on-board ship-detection suitable for small- to mid-sized satellites with low on-board processing and optical instrument performance.**

***Index Terms*—odaris, yoshi, on-board data processing, on-board data handling, image processing, ship-detection, yolo, artificial intelligence, remote sensing, system architecture, software development, quality of service**

I. Introduction

Satellite missions with focus on remote sensing typically follows the approach of downlinking the complete raw on-board imagery data to ground and afterwards filtering and processing the image data. This concept can cause latencies up to several hours before the processed image is available for further analysis, depending on the available ground infrastructure. This is critical for stakeholders interested in fast evaluation of such images to enable quick decision making in dangerous situations. With the rise of AI-models in the field of image analysis the overall process could be improved, but the initial problem of a long duration from data generation to final evaluation via the satellite communication link as bottleneck remain.

The ability to detect a critical event on-board with immediate feedback would enable the timely reaction of specific stakeholders, in turn greatly increasing the service quality. However, implementing such a real-time information system on small satellites is challenging. Suitable real-time links are not commonly available, and the AI-based on-board object detection requires advanced data processing capabilities. Nevertheless, at "German Aerospace Center" (DLR) such a real-time information system is under development; "On-board Data Analysis And Real-Time Information System" (ODARIS).

The ODARIS experiment shall provide on-board image evaluation and real-time services with an information latency around a few minutes and targets especially small- to medium-sized satellite systems. Within this software system, "Yolo-ODARIS Ship-detection Interface" (YOShI) is responsible for flexible cross-platform implementation of AI-based image analysis.

In this paper, the different modules of YOShI are presented and their effects on the images and features of interest are analysed. Subsequently, an investigation into the ship-detection performance sensitivity to processing variations is displayed and the first lab-based results presented. At the end YOShI's status and outlook towards future versions is discussed.

II. The ODARIS experiment

The development of the YOShI pipeline takes place in the context of the ODARIS experiment.

In this section a brief introduction and overview of the experiment is presented. For more detailed insight, predecessor articles about ODARIS are available: [1] and [2].

A. ODARIS Overview

The ODARIS concept originates from the demand for a real-time information system capable of providing detailed data from on-board processing within a remote system. ODARIS was designed with different key features in mind:

- **Real-time communication capability**:
  Provide bi-directional data exchange between the flight system and a user on ground.
- **Suitable as a space application service platform**:
  Enable other developers a user-friendly implementation of their own data processing applications.
- **Broad support for flight platforms**:
  Integrate ODARIS on a wide range of flight hardware and software environments.
- **Quality of services**:
  Ensure robustness and fault tolerance on long term missions without physical access to the system.

As a predecessor flight experiment, ODARIS was tested within the "Autonomous real-time detection of moving maritime objects" (AMARO) Experiment, which was performed during an aerial flight over the North Sea [3]. The objective was to detect ships, using a customized detection algorithm and a high-resolution optical camera, and to immediately inform a user on ground about the maritime situation [4]. If a ship of interest was detected, an automated alarming system was triggered to notify the experimenters in form of a short text message. The transmission of the text messages was performed via the "Short Burst Data" (SBD) Service [5] of the globally and 24/7 available satellite network, Iridium [6]. Stakeholders for such systems are official rescue and security authorities, which benefit from immediate notifications in time-critical situations.

As the next evolutional step, the ODARIS Experiment shall be performed on a satellite mission. 3 major scientific and engineering questions were defined for this mission:

- Are telecommunication satellite networks, targeted for ground-usage, applicable for real-time communication of satellites in Low Earth Orbit (LEO)?
- Is ODARIS reliably operational under space conditions?
- Is ODARIS deployable and maintainable in satellite systems?

The software system architecture can be summarised as **data-centric** and **services-based**, see Figure 1. The system consists of several services, each assigned to a specific task, that can be categorized into 3 functional areas:

- **Information System**:
  The core of ODARIS, where the automated alarming feature and custom user queries will be handled.
- **Communication System**:
  Handles all available communication channels between the on-board system and Ground.
- **Data processing apps**:
  Summary for all kind of data processing applications which are integrated in ODARIS.

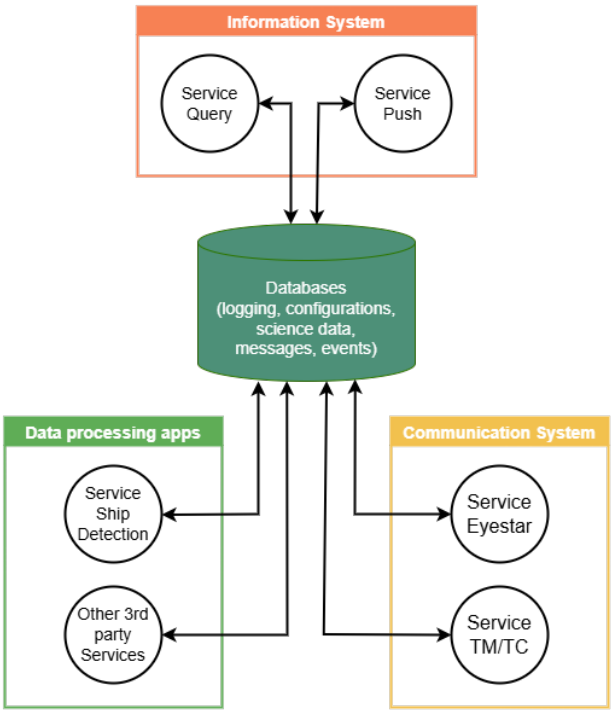


Fig. 1: ODARIS system architecture [1]

With a data-centric approach, the system becomes flexible and robust against unexpected errors during runtime. The different services are cycle-based and execute their fixed routine, while communicating with each other via a centralized database. The database is based on SQL and implemented via the SQLite C++ library [7]. An unexpected restart of a service, e.g. because of a single-event upset (SEU), doesn't affect the functionality of the system. The scientific data is stored within the common database and each service of ODARIS can therefore be turned off and on anytime during the mission.

The experiment will be performed on two upcoming flight missions, both targeted to be launched within 2026; the DLR internal "Scalable On-board Computing for Space Avionics" (ScOSA) flight mission [8], and the Seamless Radio Access Networks for Internet of Space (SeRANIS) flight mission [9] [10] managed by the University of the Bundeswehr Munich [11].

The satellite missions have common technical constraints:

- Limited RAM available for image processing
- High "Ground sampling distance" of LEO satellite images
- Limited bandwidth for in-orbit file uplinks

The ScOSA system provides several "High Performance Nodes", where each node is based on the AMD Zynq-7020 SoC with a dual core ARM Cortex-A9 processor [12]. The total available RAM on a single node is 512 MByte, while the available amount for the ODARIS image processing is estimated to be around 256 MByte. On-board, the payload camera employed is the Simera TriScape50 [13]. This is a RGB snapshot camera in the visible spectrum with a RGGB

Bayer format colour filter array (CFA). The imager has a resolution of 4096×3072 pixels, resulting in a "Ground sampling distance" (GSD) of 30m/px at an orbital altitude of 500km.

In this paper, the technical constrains from the ScOSA flight mission will be used to describe the concept of YOShI.

B. YOShI in ODARIS

YOShI was created to enable state-of-the-art object-detection with limited system resources. To reduce processing demands, the main focus is on discarding unnecessary data from the unprocessed satellite images and enhancing the presence of features of interest. The feature enhancement has been developed in terms of maximizing the value of the data within the images. As an object-detection AI-model, YOShI integrates a "You-Only-Look-Once" (YOLO) model [14], which is customised in accordance to the available storage capacity and the computational processing power within the satellite missions of the ODARIS experiment.
Overall, the pipeline can be separated to 3 major parts:

- Image pre-processing
- AI-inference
- Image post-processing

The image pre-processing part prepares the camera image for the AI-inference and filters out unnecessary sections, like cloud coverage or land masses. Inference is performed with a customised YOLO model, POOCHY, aiming to detect larger ships on single tiles of the satellite image. As on-board runtime environment Google's RTLite, former TFLite, is used [15]. In the post-processing, the inference results are formatted and stored into a database, accessible for other ODARIS services. A full overview of the YOShI pipeline and its single steps can be seen in Figure 2.

There are several technical aspects, that YOShI needs to fulfill for the flight mission:

- Keeping the RAM usage within the available limit during the image analysis.
- Shorten the overall processing time, to keep the system feasible for low-latency data provision.
- Reduce the size of the AI-model file, to enable in-orbit updates with limited bandwidth.
- Maximize the object-detection quality of the customized AI-model.

These different aspects often conflict with each other. Therefore, an iterative balancing via fine tuning of the YOShI pipeline parameters had to be performed.

Within the next sections, section III and section IV, the components of the YOShI pipeline are described in detail, as well as the customized ship-detection AI-model POOCHY. All technical details presented apply to the ScOSA flight mission implementation.

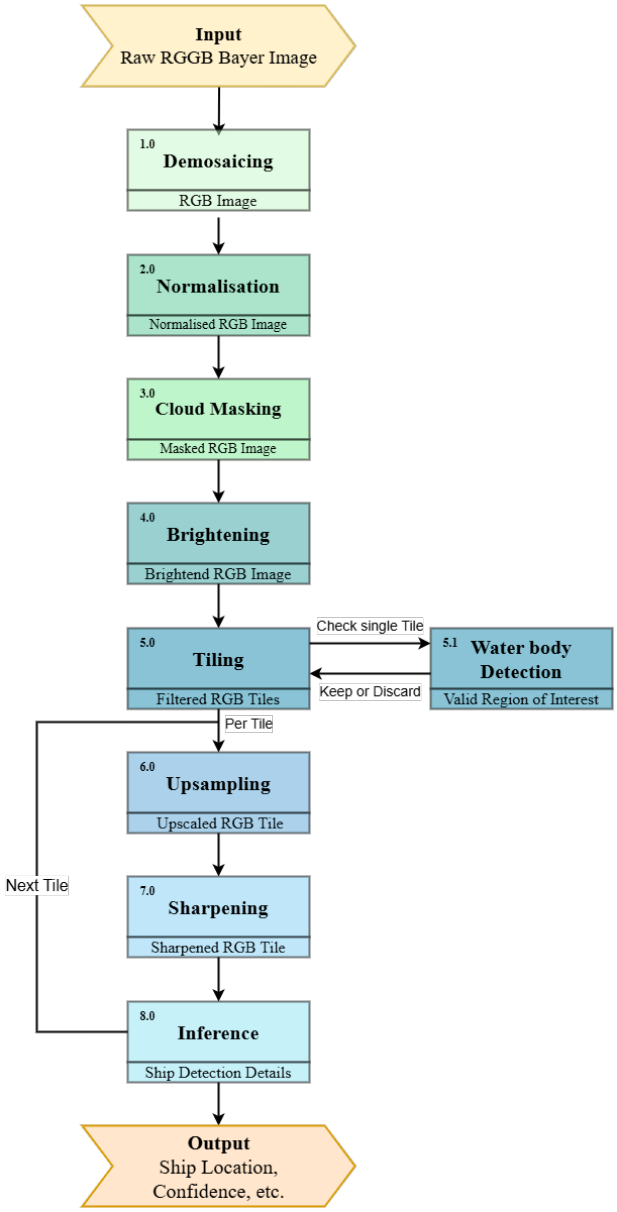


Fig. 2: YOShI pipeline overview

III. YOShI Processing Pipeline

The *TriScape50* imager produces 4096 by 3072px RGGB Bayer snapshot images [13], which after demosaicing total to 37.75 MB in size. Not only is the size of this image challenging to train an AI-model on, but running inference on an image of this size would fill up the entire available RAM of the on-board computer.

YOShI is a dual-purpose processing pipeline, developed for C++ environments. First, it aims to efficiently manage the image data from the TriScape50 imager to POOCHY, the YOLO-based AI-model. Secondly, a modular set of pre-processing steps in YOShI aims to increase the quality of the images to aid in the ship-detection efforts. As a result of the computational restrictions, YOShI was developed with simplicity and modu-

larity at its core. Therefore, YOShI integrates a tiling approach to the large image problem, splitting the processing into two stages:

1) Core processing applied on the entire image
2) Further tile-based processing only for regions of interest

At the begin of the image processing pipeline YOShI performs the demosaicing of the raw *TriScape50* data. This stage transforms the one-dimensional raw sensor data into a three-dimensional RGB image by inferring the remaining pixel values of each band from the neighbouring values. For this process OpenCV's colour space conversion algorithm `BayerRGGB2RGB_EA` is applied [16]. The edge-aware version of the function is used, because the low resolution of 30m/px leads to even large container ships appearing relatively small in the overarching image. The added computational cost of the edge-aware function comes with a stronger edge-preservation, which is crucial for object detection models like YOLO that focus on object shape and contour.

Experimental results suggest the bit depth of images have a negligible effect on the ship-detection performance of YOLO models. In accordance with this, the 10bit images of the *TriScape50* are reduced to 8bit after demosaicing, to ease the computational load. This step is referred to as the *Normalisation* step in Figure 2.

Followingly, the next core processing step is cloud masking. This is performed image-wide to encompass all cloud instances and mask these effectively. Applying this step tile-wise could lead to partitioned cloud instances that would fail the minimum area threshold for a cloud, hindering the cloud masking. Colour thresholding and morphological operations are used to mask cloud instances that meet the area threshold. Because cloud masking could also mask ships underlying the clouds, the masking threshold is set such that thin cloud covers that would permit ship-detection are not masked. While cloud-masking helps with computational load, and reduces the probability of false positives of the AI-model, the main purpose of cloud masking is to enable the processing of only ground-relevant pixel values.

With this in mind, the subsequent core process of YOShI is the brightening step. Here, the overall exposure of the image is corrected. The three bands of the masked image are stacked into one channel, the luminosity, which is then brightened through an alpha-beta linear contrast adjustment. Without cloud-masking, the contrast between the clouds and ocean background would overexpose the image, leading to indiscernible image instances. Therefore, the linear contrast adjustment of the ground-relevant pixels helps highlight the ship contrast from the ocean background, leading to improved ship-detection potential.

Before progressing to the tiling process, the concept of discrimination function must be introduced. In this context, the term discrimination function is used to indicate any non-AI methods, that are used to identify whether a tile potentially contains objects of interest. For image classification a wide range of discrimination functions exists, including anomaly detection methods, like " Principal component analysis" (PCA) approaches [17], as seen in [18]. In general, they all follow the same core concept; saving computational time by discarding tiles, that are not relevant for further investigation.

For YOShI the discrimination function is based on waterbody detection, which checks if the tile contains an area of water that is large enough to contain a ship. This approach mimics the colour thresholding of cloud masking on shades of blue, and maintains an area threshold for sufficiently large bodies of water. As a result, computational time and processing is saved on tiles that predominantly contain land. This approach allows all tiles, which contain waterbodies, to be processed by the AI-model, without omitting any ship instances.

The waterbody detection discrimination function is used concurrently with the tiling processes, as illustrated in Figure 2. The satellite image is scanned through tile-by-tile, and waterbody detection is applied per tile. The tile size was empirically set to 256×256 squared tiles, as this struck a balance between number of tiles and probability of partitioning a ship over several tile instances. The 4096×3072 satellite image, thus yields a total of 192 non-overlapping 256×256 square tiles. However, during the scanning, waterbody detection dictates whether the tile is to be kept or discarded. Therefore, the output of this processing step is a satellite image containing only the tiles satisfying the waterbody criterion.

The following tile-processing step is tile upsampling, or also referred to as upscaling. This processing step uses OpenCV's `resize` function with Lanczos as interpolation method to synthetically increase the dimension and resolution of the image. Upscaling adds no new "ground-truth" data to the image, instead, the additional data is interpolated from surrounding pixels. Nevertheless, upscaling increases the number of pixels per ship instance, delivering more data to an AI model to base its detection on. As with the demosaicing process, the more computationally intensive interpolation algorithm, Lanczos, was chosen due to its edge preservation for ship instances. The performance of Lanczos as upscaling algorithm has been found to be the most suitable in ship-detection contexts, as seen in [19]. For the ScOSA flight mission, a $2\times$ upscaling is performed on the tiles, changing their size from 256×256 to 512×512. This is synthetically increases the resolution of the images from 30m/px to 15m/px.

The final step before passing the upscaled tile to POOCHY involves a sharpening step. The synthetic resolution augmentation step of Lanczos upscaling can yield blurry images, of which definition could be increased for the AI-model. Therefore, sharpening was tested on sample images, and gains were found primarily on the detection confidences. Derived from experimental testing, it was found, that sharpening through unsharp masking yielded the best results from the sharpening methods, despite the images appearing noisy to the human eye.

After the pre-processing phase of the YOShI pipeline, the image tile is passed to POOCHY for inference. Details about the AI-model can be found in section IV. The results of the inference are stored in the ODARIS on-board database, accessible for the automated alarming system of ODARIS and for stakeholders on Ground.

The tile-based pre-processing, inference and post-processing is performed iteratively, until every relevant tile has been analysed.

IV. The POOCHY model

A survey of possible open-source CNN models was performed, with the following criteria:

- Suitable for object-detection on images
- Providing fast results to enable real-time information of ODARIS
- A small model file size for potential in-orbit updates

As a results the YOLO model was chosen for the ship-detection. Compared to other AI-models, the YOLO models were found to strike the best balance of speed and performance, compared to other single-stage detectors [20] [21].

From the available YOLO models, the newest YOLO-model version "YOLOv11" by Ultralytics [14] was chosen, because it provides higher performance at smaller model sizes compared to other more established models like YOLOv7 or YOLOv5.

The aforementioned strict mission requirements derive from the challenges of in-orbit use of AI-models. The key issue is the availability of useful datasets for training AI-models on-ground before their in-orbit use. Within the scientific community, investigative paths diverge on this topic. Some choose self-supervised AI-models that can be re-trained in-orbit with on-board data, while other options involve the use of pre-trained models for in-orbit re-training. However, in-orbit retraining needs high computational power on the on-board hardware. For YOShI a highly streamlined lightweight model was used, enabling model re-upload over S-band, and leveraging on-ground training infrastructure.

The architecture of the YOLO11n was surveyed for potential size-savings applicable for the ScOSA flight mission. The backbone, neck and head of the YOLO11n model are setup to detect object instances at different spatial resolutions within the image. More specifically, YOLO11n uses three detection heads at different scales - P3 (small), P4 (medium), and P5 (large) [22]. Therefore, the heads and their respective backbone parts complement each other's detection scales during inference.

However, the effective dimensions of the ships in remote sensing images can be estimated through the optical instrument resolution and orbital altitude. With a GSD of 30m/px, the average cargo ship would appear as an object with the dimensions of 10 by 2 pixels. This means that after image pre-processing, the YOLO model needs to detect ship instances of size $\approx 10 \times 2$ pixels within 256×256 image tiles. Consequently, any part of the YOLO model architecture that focusses on objects beyond this size is not applicable and can be discarded. After pruning of both the backbone and head of the model, the remaining model architecture contains 4 layers in the backbone and one detection head. The components of this pruned model can be seen with respect to the original YOLOv11 model in Figure 3. This pruning cut down the model size from the 5.5 MB (`.pt` format) down to 813 kB (`.pt` format). The detection performance of the model improved due to the focus on a singular class of object size. The model sizes do not possess any form of further quantization.

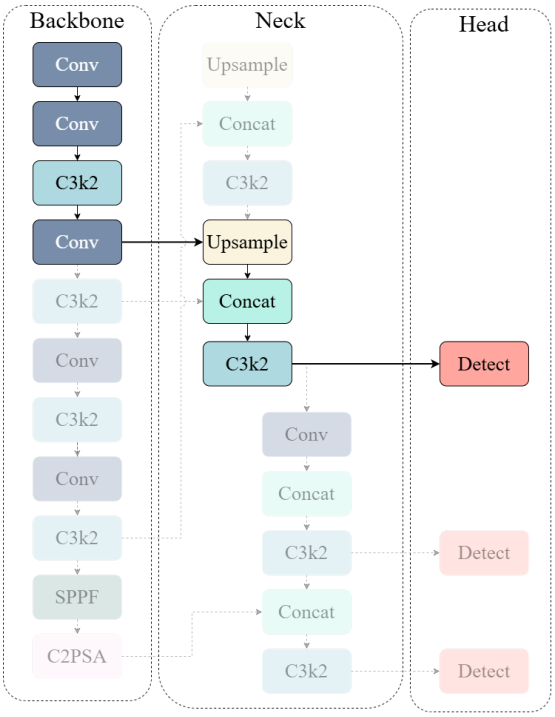


Fig. 3: The POOCHY model components extracted from the original YOLOv11 model architecture of [14].

The POOCHY model training was performed in a python environment using the Ultralytics library [14]. Moreover, POOCHY was trained and validated on an independently created database of Sentinel-2 RGB images, re-sampled to a GSD of 30m/px. The database includes 2612 ship instances from 97 Sentinel-2 images and in 52 different geographical locations across the globe.

The chosen metric to evaluate AI-model performance in this paper is the F1 score. The F1 score consists of the harmonic mean between precision and recall of an AI-model, and is defined by Equation 1. Precision focusses on the correctness of the detections, while recall targets the detection coverage. The F1 score balances the number of false positives (incorrect predictions) with the number of false negatives (missed predictions) of a model. The metric of evaluation of an AI-model is dependent on the requirements of the application. Here, to evaluate the POOCHY models' performance the F1 score was chosen.

$$\mathrm{F1} = 2 \cdot \frac{\mathrm{Precision} \cdot \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}} \tag{1}$$

Where precision and recall are defined as:

$$\mathrm{Precision} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FP}} \qquad \mathrm{Recall} \qquad = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}}$$

Note: TP = True Positives, FP = False Positives, FN = False Negatives.

V. Results

The custom YOLO model, POOCHY, was tested under different processing conditions to encapsulate the potential of the model for the ODARIS missions. The results of this testing can be found below, starting with Table I.

TABLE I: Results overview of the POOCHY model performance across upscaling levels.

Upscale level	Synthetic GSD [m/px]	Model size (.tflite) [kB]	RAM_max [MB]	Inference time [ms]	F1 score & confidence level
1x	30	1464	13	1175	0.688 / 0.314
2x	15	1536	43	4626	0.709 / 0.312
3x	10	1656	95	10576	0.655 / 0.296
4x	7.5	1824	160	19125	0.700 / 0.285

Inspecting Table I it is visible that the relation between F1 score and upscaling level is similar to parabolic, with a local peak at $2\times$ upscaling. This peculiar result indicates that the synthetic augmentation of the effective GSD, is not always beneficial to the ship-detection performance of the AI-model. The maximum performance is extracted when using the $2\times$ upscaling, reaching a max F1 score of 0.709. Though upscaling at $4\times$ increases performance again to 0.700, it does not out-perform the $2\times$, despite the twice as high effective resolution. The Confidence-F1 curves of all the different upscaling levels can be seen in Figure 4, where the curves display the behaviours seen in Table I.

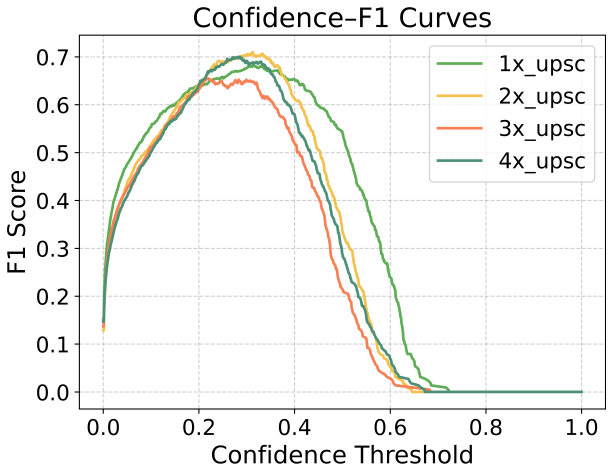
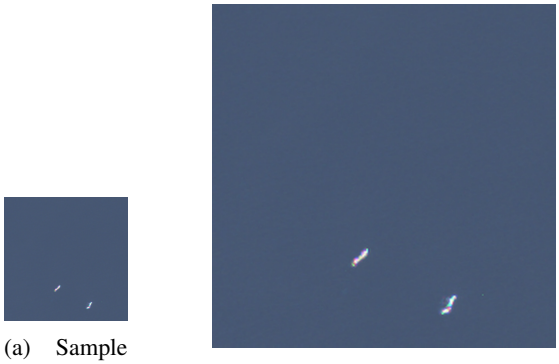


Fig. 4: The Confidence-F1 curves of the different upscaling levels

To better understand the limited performance gains from higher levels of upscaling, Figure 5a and Figure 5b were included. Here the limitations of the upscaling algorithms start to appear to the human eye. The interpolation algorithms used in the upscaling, even when advanced like the Lanczos, do not produce ground-truth information, rather these interpolate data from neighbouring pixels. Therefore, the resulting images appear blurry, and artifacts are even introduced in the images, as can be seen on the lower side of the bottom ship in Figure 5b. Consequently, the amount of data passed to the AI-model increases with increased upscaling, but with increased



(a) Sample $1\times$ upscaled image with a GSD of 30 m/px.

(b) Sample $4\times$ upscaled image with a synthetic GSD of 7.5 m/px, here scaled approximately with respect to $1\times$.

Fig. 5: Comparison of $1\times$ and $4\times$ upscale level image tiles..

upscaling comes added blurring, larger artifacts, and noise amplification. With this in consideration, it is not surprising that an optimum balance is found then with $2\times$ upscaling. At this stage, the added data in object shape and definition is valued by the AI-model, while the noise amplification remains moderate.

The blurring effect is also reflected by the results in Figure 4. More specifically, the trend of decreasing confidence threshold with increasing upscaling level. The lower the confidence threshold of an AI-model, the more low-confidence detections are permitted by the model, while maintaining performance. This is a positive attribute of a model, as it is closely related to the generalisation ability of a model. The added blurring of high upscaling levels decreases the detail within ship instances, reducing the overall F1 score, but, in turn, homogenising the appearance of ship instances.

Nevertheless, for the ODARIS mission, the selected upscaling level and corresponding model, is the $2\times$ level. Not only is this the model with the highest performance, it is also has a more restricted model size in `.tflite` format compared to the higher upscaling levels. The $2\times$ model size is within 5% of the model size of the $1\times$ model, which is key when considering the in-orbit re-upload philosophy employed for the ScOSA mission. Moreover, the increased computational time of the $2\times$ upscaling can still be justified in the long-term. On the other hand, $3\times$ and $4\times$ inference times are high, and are not compensated for with F1 score performance.

VI. Conclusion and Way Forward

In this article both the "Yolo-ODARIS Ship-detection Interface" concept and the custom AI-model, POOCHY, were presented. YOShI demonstrates itself as a simple, modular, and flexible approach to on-board pre-processing of images for on-board AI model deployment. The processing includes image-wide processing elements; demosaicing, normalisation, cloud masking, brightening. This is followed by a concurrent tiling and waterbody detection step, to discard tiles without potential for ships. Finally, tile-based processing is conducted

with the processing steps of; upscaling, sharpening, and finally inference.

The customisation of POOCHY from YOLOv11 follows from knowledge of the appearance of ship instances in remote sensing images. POOCHY employs 4 backbone layers and one detection head, yielding model sizes smaller than 1.9MByte for upscaling less than $4\times$.

The lab-based results of YOShI and POOCHY, shown in Table I, indicate that the ideal ship-detection performance is obtained at $2\times$ upscaling with an F1 score of 0.709. However, this comes at the cost of nearly 3 times the maximum ram usage (43MByte), and nearly $4\times$ the inference time (4626ms). For the ScOSA flight experiment, the $2\times$ upscaling level was selected.

Beyond the iterative parameter tuning of YOShI and POOCHY to best fit the TriScape50 specifications, work was invested into further generalising the applicability of YOShI and POOCHY. As a result, a training pipeline was built. This infrastructural tool enables rapid prototyping of on-board processing and POOCHY variants tailored to ship-detection, with compatibility to optical instruments beyond the *TriScape50*. As a next step, the flexibility of YOShI shall be proven by implementing it on the upcoming SeRANIS satellite mission, which has a different on-board camera and computer architecture. Additionally, the infrastructure for AI-model training and deployment will be extended to support different kinds of on-board computer architectures and Inference-frameworks.

References

[1] Daniel Herschmann and Kurt Schwenk. On-board data analysis and realtime information system - software development concept for new space. In *Deutscher Luft- und Raumfahrtkongress 2023 (DLRK 2023)*, November 2023.

[2] Kurt Schwenk and Daniel Herschmann. On board data analysis and realtime information system. In *Proceedings of the 2023 European Data Handling and Data Processing Conference for Space, EDHPC 2023*, 2023.

[3] Katharina Willburger, Kurt Schwenk, and Jörg Brauchle. Amaro - an on-board ship detection and real-time information system. *Sensors*, 20(5):1324, Februar 2020.

[4] Kurt Schwenk, Katharina Willburger, and Sebastian Pless. Amaro-autonomous real-time detection of moving maritime objects: introducing a flight experiment for an on-board ship detection system. In Ulrich Michel and Karsten Schulz, editors, *Earth Resources and Environmental Remote Sensing/GIS Applications VIII*, volume 10428, pages 71 – 81. International Society for Optics and Photonics, SPIE, 2017. DOI: 10.1117/12.2277991.

[5] Iridium Communication Inc. Iridium sbd, 2025. https://www.iridium.com/services/iridium-sbd/.

[6] Iridium Communication Inc. Iridium-llc home page, 2025. https://www.iridium.com.

[7] SQLite Project. Sqlite project homepage, 2025. https://sqlite.org/.

[8] Daniel Lüdtke, Thomas Firchau, Carlos Eduardo Gonzalez Cortes, Andreas Lund, Ayush Mani Nepal, Mahmoud Mostafa Hussein Hassan Elbarrawy, Zain Alabedin Haj Hammadeh, Jan-Gerd Meß, Patrick Kenny, Fiona Brömer, Michael Mirzaagha, George Saleip, Hannah Kirstein, Christoph Kirchhefer, and Andreas Gerndt. Scosa on the way to orbit: Reconfigurable high-performance computing for spacecraft. In *2023 IEEE Space Computing Conference, SCC 2023*, August 2023.

[9] Johannes Bachmann, Artur Kinzel, Francesco Porcelli, Alexander Schmidt, Robert Schwarz, Christian Hofmann, Roger Förstner, and Andreas Knopp. Seranis: In-orbit-demonstration von spitzentechnologie auf einem kleinsatelliten. In *Deutscher Luft- und Raumfahrtkongress 2022, Dresden*. Deutsche Gesellschaft für Luft- und Raumfahrt - Lilienthal-Oberth e.V., 2024. DOI: 10.25967/570020.

[10] SeRANIS Redaktions-Team. Seranis public website, 2025. https://seranis.de/.

[11] UniBw München Redaktions-Team. University of the bundeswehr munich public website, 2025. https://www.unibw.de/home-en.

[12] Advanced Micro Devices, Inc. Zynq-7000 soc product description, March 2025. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[13] Simera Sense. Triscape50 camera product page, 2025. https://simera-sense.com/products/triscape50/.

[14] Ultralytics. Ultralytics homepage, 2025. https://opencv.org/.

[15] Google Inc. Tensorflowlite project homepage, 2025. https://ai.google.dev/edge/litert.

[16] OpenCv. Opencv project homepage, 2025. https://opencv.org/.

[17] OpenCv. Introduction to principal component analysis (pca), 2025. https://docs.opencv.org/4.x/d1/dee/tutorial_introduction_to_pca.html.

[18] Tao Zhang, Armando Marino, Huilin Xiong, and Wenxian Yu. A ship detector applying principal component analysis to the polarimetric notch filter. *Remote Sensing*, 10(6), 2018. ISSN: 2072-4292. DOI: 10.3390/rs10060948.

[19] Ch Muhammad Awais, Marco Reggiannini, and Davide Moroni. Image quality vs performance in super-resolution for sar ship classification. In *2025 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1–5, 2025. DOI: 10.1109/ISCAS56072.2025.11043629.

[20] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[21] Nidhal Jegham, Chan Young Koh, Marwan Abdelatti, and Abdeltawab Hendawi. Yolo evolution: A comprehensive benchmark and architectural review of yolov12, yolo11, and their previous versions, 2025. https://arxiv.org/abs/2411.00201.

[22] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements, 2024. https://arxiv.org/abs/2410.17725.