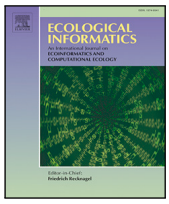
Ecological Informatics 93 (2026) 103569

Contents lists available at ScienceDirect

Ecological Informatics

journal homepage: www.elsevier.com/locate/ecolinf



Towards decentralized ecological monitoring: A local-first web-based AI platform for sea turtle detection

Alessio Pierluigi Placitelli [a,b,*], Sandra Hochscheid [c], Fulvio Maffucci [c], ALan F. Rees [d], Gianluca Treglia [c], Antonino Staiano [b]

[a] Department of Biology, University of Naples Federico II, Via Cinthia, 26, Naples, 80126, Italy
[b] Department of Sciences and Technology, University of Naples Parthenope, Centro Direzionale Isola C4, Naples, 80143, Italy
[c] Department of Marine Animal Conservation and Public Engagement, Marine Turtle Research Group, Via Nuovo Macello 16, Portici, 80055, Italy
[d] ARCHELON, the Sea Turtle Protection Society of Greece, Solonos 113, Athens, 10678, Greece

ARTICLE INFO

Keywords:
Sea turtle monitoring
UAV imagery
Artificial intelligence
Environmental software
Browser-based applications
Wildlife detection
Client-side processing

ABSTRACT

Accurately monitoring sea turtle populations is crucial for informing effective conservation strategies; however, traditional methods for assessing spatial distribution and abundance are time-consuming, labour-intensive, and prone to observer bias. Unoccupied Aerial Vehicles (UAVs) have become valuable tools for collecting high-resolution imagery with minimal disturbance to marine fauna. However, the lack of standardized and user-friendly platforms for processing UAV data limits their broader application. In this paper, we present a lightweight browser-based web application designed to streamline the analysis of video collected by UAVs to monitor sea turtles. The application requires no additional software installations beyond a modern web browser and operates entirely on the client side, preserving data privacy. It supports the integration and execution of artificial intelligence models locally, facilitating the automated detection and classification of turtles in video footage. This tool, freely available, bridges the gap between UAV data collection and effective conservation-oriented decision-making by enabling rapid, standardized, and scalable analysis. Our approach promotes community-driven development and the reuse of AI models, making environmental monitoring practices more accessible and collaborative.

1. Introduction

Generating data on the spatial distribution and population trends of marine species, such as sea turtles, is challenging, yet necessary to understand habitat use and inform conservation efforts. Indeed, estimates of sea turtle abundances and trends are key parameters for the assessment of their conservation status, yet obtaining these estimates in marine foraging areas is extremely challenging because of the elusive nature of sea turtles and the limited accessibility of these areas (Rees et al., 2016). In this context, unoccupied aerial vehicles (UAVs) have become increasingly valuable tools to assess the distribution and abundance of sea turtles in various environments, including nesting and foraging areas (Sykora-Bodie et al., 2017; Rees et al., 2018; Robinson et al., 2020; Dickson et al., 2022). In contrast to traditional methods, UAVs offer advantages such as reduced time to accurately identify turtles compared to survey on boats (Yaney-Keller et al., 2021). Compared to aircraft surveys, UAVs offer finer spatial and temporal resolution, potentially lower costs and the ability to collect data with less disturbance to animals (Pedrazzi et al., 2025; Duporge et al., 2021; Bevan et al., 2018; Papazekou et al., 2024). These aerial surveys using drones can acquire large amounts of imagery and video, which can then be analysed to identify, count, and monitor sea turtles. Typically, the analysis of these video recordings involves the review of multiple independent observers to minimize biases such as perception errors (false negatives) and misidentification (false positives) (Agabiti et al., 2024). For example, studies often employ two or three reviewers who independently watch the footage and mark turtle sightings, with discrepancies resolved by a third reviewer or through a consensus process. This manual review process, while crucial for accuracy, can be time-consuming.

However, the integration of neural networks and artificial intelligence algorithms is enhancing the automated detection and classification of turtles in UAV imagery, allowing for more rapid and efficient analysis of large datasets. Convolutional neural networks (CNN), fed

* Corresponding author at: Department of Biology, University of Naples Federico II, Via Cinthia, 26, Naples, 80126, Italy.
E-mail address: alessiopierluigi.placitelli@unina.it (A.P. Placitelli).

https://doi.org/10.1016/j.ecoinf.2025.103569
Received 5 September 2025; Received in revised form 15 December 2025; Accepted 15 December 2025
Available online 20 December 2025

with overlapping tiles from captured images, have shown promising results by detecting more sea turtles compared to manual efforts (Gray et al., 2019). Further explorations tested their ability to distinguish different marine species by analysing salient image parts (Dujon et al., 2021). Recent research, with the advent of You Only Look Once (YOLO) (Redmon et al., 2016) object detection real-time architectures, has focused on the automatic counting of green turtles using YOLOv7 (Wang et al., 2023) coupled with BoT-SORT tracking, achieving low false-positive rates in complex coastal areas (Noguchi et al., 2025).

Studies have shown (Axford et al., 2024) that AI-powered systems can significantly reduce the labour required for video analysis, additionally highlighting that open collaboration could foster significant progress in the field. While challenges such as correcting for availability bias (the underestimation that occurs when turtles are present but not visible for detection) remain important considerations, UAVs coupled with automated video analysis represent a significant advancement in our ability to understand sea turtle ecology and support effective conservation strategies (Gonzalez Nunez et al., 2024). The development of open-access training datasets for AI models could further accelerate the creation and improvement of these automated systems for global sea turtle monitoring. Despite these advances, many of the workflows developed to date still require the local installation of specialized software, often relying on outdated libraries or dependencies, and involve manual preprocessing of the input videos before analysis can begin. Such technical barriers can hinder the adoption of these tools by conservation practitioners who may lack advanced computational expertise, thereby limiting their practical impact and wider dissemination. Recent efforts to reduce manual intervention through fully automated workflows (Dimauro et al., 2022) have improved accessibility and usability for conservation practice, though a degree of manual processing is often still required. Front-end deep learning web applications (Goh et al., 2023), which deploy models directly on the client without specialized inference back-ends, promise to alleviate some of these technical barriers, albeit at the cost of a more computationally constrained environment. Nevertheless, their potential in an ecological context remains largely untapped. Recent research has focused on using the web to access back-end processing power (Subeesh et al., 2024; Zhou et al., 2025; Berger-Wolf et al., 2017) or to visualize pre-computed data (Martínez-Movilla et al., 2024).

Building on this, our paper introduces a novel web application designed to streamline the analysis of data collected by UAVs for the monitoring of sea turtles. In its primary operating mode, no third-party software is required beyond a modern web browser, making it accessible to a broad range of users. Furthermore, the web application does not interact with any back-end services besides the one used to serve the web page. Such a tool is needed to bridge the gap between the increasing availability of UAV technologies and the lack of standardized, user-friendly platforms for processing population trend data collected by UAVs. Additionally, we present two original sea turtle detection datasets that the research community can use and share for research purposes. These datasets are still being populated with newly acquired videos, as UAV harvesting campaigns are time-consuming and resource-intensive.

We hope that our work will provide a foundation and a common framework that sea turtle researchers can use to share their AI models with the wider community. The web-application is available for experimentation at https://www.a2p.it/seaturtles. Its source code is hosted on GitHub at https://github.com/neptun-ia-lab/turtle-det-web, where any researcher can contribute their own datasets, case studies and models. From the web-application, models can be executed locally by opening recorded footage on local machines without uploading it to third-party services. This provides researchers with immediate insights into sea turtle populations and behaviours, enabling them to develop more effective conservation strategies by quickly evaluating the distribution and abundance of sea turtles. Our goal is to improve the efficiency of sea turtle monitoring, reduce the need for manual data analysis, and minimize disturbance to the animals. This will support better informed decision-making and improved conservation policies. In this era of technology-enabled conservation, continued innovation and collaboration are crucial to ensuring the long-term survival of vital marine species, in line with the Sustainable Development Goal (SDG) 14 (*Life Below Water*)[1] of the United Nations SDGs 2030 Agenda, which aims to conserve and use the oceans, seas, and marine resources sustainably.

2. Architecture overview of the web application

Inspired by the principles of local-first software (Kleppmann et al., 2019), we designed a web application that allows researchers of the marine ecosystem to investigate sea turtles within a local session on a web browser page, as depicted in Fig. 1. This is achieved by downloading computer vision models and executing them in any sufficiently modern web browser with WebGPU support using Web Platform APIs, on a machine with adequate hardware. By doing this, researchers can utilize their local resources without resorting to a cloud environment or uploading their recorded UAV footage to third-party servers. Furthermore, in its self-contained in-browser version, the application does not require the installation of any third-party software or dependencies, as it is exclusively developed using the capabilities of the Web Platform.

The web application is based on the Full Stack FastAPI Template (Ramírez, 2025) and is composed of two main parts, a back-end and a front-end. The back-end is written in Python and provides core functionalities for user-management and storage. Though included in the used template, these capabilities are not used in the present version. For this reason, the rest of this section will focus exclusively on the front-end part, leaving back-end improvements for future work [6]. The front-end part, what the user directly sees and interacts with in the web browser, is written in TypeScript to increase its robustness and reduce potential bugs, primarily due to the type checks enabled by using this language (Gao et al., 2017). The User Interface is developed using ReactJS and Chakra UI libraries.

A Processor is the fundamental building block of the developed application. It is fed with raw image data (for example, RGB frames of the video being played) and returns the results of the processing, in the form of detected object labels, the detection confidence scores, and their location within the raw image as axis-aligned bounding-box coordinates.

Versioning follows the GitHub commit history, where the latest commit's revision hash identifies the release. In the current software version (`d46ac7e`), the web application ships with two types of processors:

- **in-browser processors**; in this processor type, video frames are processed using the onnxruntime-web (ONNX Runtime developers, 2018) library. No additional software installation is required, as the processor works exclusively with standard features of the web platform. When the processor is initialized, it downloads any resource it needs to run locally within the browser session, then it warms up by executing a few inference calls with random inputs. Some models might be limited or unsupported, depending on the chosen execution provider[2] and the maturity of the browser support;

[1] https://www.globalgoals.org/goals/14-life-below-water/.

[2] Execution providers allow executing ONNX graphs using specific hardware or software capabilities, e.g., WebGPU, Wasm. While Wasm inference is universally supported across all modern browsers, its performance are not as good as WebGPU, though the support for the latter is still not mature.

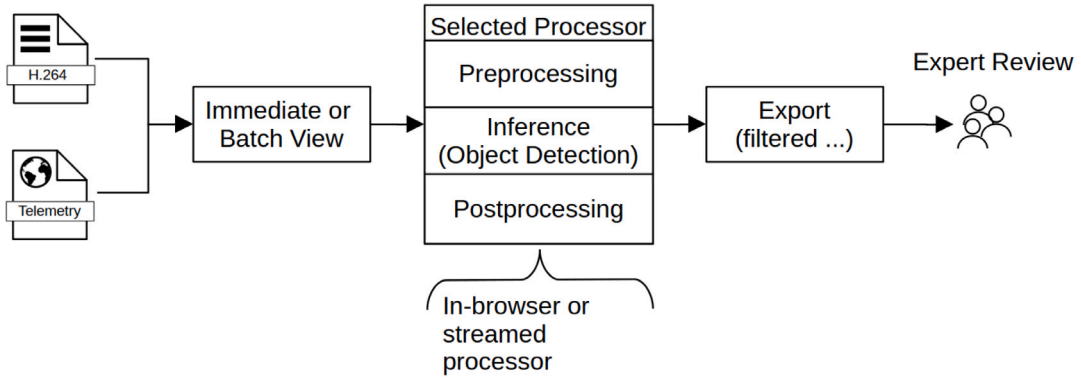


Fig. 1. A high-level overview of the analysis workflow.

- **streamed processors**; in this processor type, enabled in the web-application by selecting the "HTTP Streaming" model type, the video frames are streamed to a back-end service written in Python, which executes a pipeline equivalent to the one implemented via onnxruntime-web, but directly using the local hardware resources via the chosen processing libraries (e.g., CUDA). Processors of this type are meant to overcome the limitations that might arise with in-browser processors, for example, memory restrictions or unimplemented ONNX operators. The back-end service can run on any reachable network machine, including localhost. Due to resources considerations, our platform does not currently provide publicly available hosted back-ends. However users can create and host them based on the templates available on our GitHub repository.

As new state-of-the-art architectures become available, supporting them in the web application requires implementing a new processor. The next section will focus on the typical inference workflow that is common to both processor types.

2.1. The inference workflow

The platform currently supports two operating modes, which correspond to different UI views: immediate (Section 2.2) and batched (Section 2.3). For this overview, the difference between the two modes is negligible, as the only difference is in the pace at which they can process the incoming data and in the way the results are presented.

The workflow starts with the platform user opening a video file in MP4 format (Lim and Singer, 2006), containing the footage to be analysed. If the footage is recorded using a UAV and its flight telemetry is available, it can be found within the video file's subtitle track or as a separate subtitle file. Due to the limitations of the browser environment and the libraries we use, the telemetry embedded in the video file cannot be used directly; it must be loaded as a separate SRT file. Making the flight telemetry available enables the application to associate latitude and longitude coordinates with the detections, allowing it to estimate the size of the detected bounding box.

Given the detected bounding box B, its width B_w and height B_h are estimated by the model in pixel units in the image space. These dimensions can be estimated in metres using the ground sampling distance (GSD). The GSD_w for the width can be calculated as

$$gsd_w = \frac{h \cdot s_w}{f \cdot i_w}$$

where h is the altitude of the UAV, taken from flight telemetry at the time the frame was recorded. f is the actual focal length of the camera, s_w is the width of the camera sensor, and i_w is the width of the frame. All of these values are specified in the application settings. The GSD_h for the height can be computed similarly by replacing the sensor width and frame width with the respective heights. The dimensions of the detection bounding box can be expressed in metres as $(B_h \cdot GSD_h, B_w \cdot GSD_w)$.

The platform initializes the active processor, selected by the user in the settings panel, by fetching any required resources and performing a warm-up using random data. Once the processor is ready, the video frames begin to be processed. To reduce the time taken to run inference on a full video, frame-skipping settings are used to determine whether the current frame needs to be skipped or can proceed to the next stage of the pipeline.[3] From this point onward, the frames are processed as individual raw images.

The pre-processing, inference, and post-processing stages are applied to each frame. The pre-processing stage prepares the raw images to fit the expectations of the inference model. For example, if the model requires the images to be a certain size, they are resized. The pre-processed frame is then handled by the processor in the inference stage, which sends the data to the wrapped model and returns the results when they are available. Finally, the post-processing stage interprets the model output, providing consistent data in the form of confidence scores, bounding boxes scaled to the original image size, and labels. These results are then combined with those from previously processed frames and presented on the screen.

2.1.1. Pre-processing as part of the model

In a typical pipeline, pre-processing operations such as image resizing are performed before the data is fed to the model. In the context of web applications, these operations are usually carried out using *JavaScript/TypeScript* and the Web Platform Canvas API. For example, resizing an image requires an *OffscreenCanvas* to be created, the image content to be copied to it at the new size, and then the raw image data to be retrieved and fed to the model. In order to reduce inference time and make the pre-processing phase more efficient, these operations can be expressed as a model. Our pipeline for doing so looks as follows:

- We redefine any pre-processing operation using the torch library in Python (see algorithm 1).
- The implemented pre-processing model is then exported to an ONNX file using the torch APIs.
- The generated model can be either fused with the main inference model or kept separate. With the former, the input is fed to a single fused model that provides the caller with the inference results. With the latter, the pre-processing model must be queried, and its output used as input for the inference model, which is then fed to the post-processing model. The implemented processors use fused pre-processing mode.

[3] E.g., if frame-skip is set to 3, a frame every 3 is processed, saving computational resources.

This enables the preprocessing, inference, and post-processing models to be pipelined within ONNX, thereby reducing the number of copies between the GPU and CPU (and vice versa) and improving the overall inference performance. It also enables a higher inference rate to be achieved in both the immediate and batched views. Additionally, the same preprocessing pipeline is used for both streamed and in-browser processors, ensuring consistent behaviour across processing modes.

Algorithm 1: SAHI preprocessing

Input: Image I, slice size $s_w \times s_h$, overlap ratio r
Output: Set of sliced image patches S
Initialize $S \leftarrow \emptyset$;
Obtain image width W and height H;
for $x \leftarrow 0$ ***to*** W ***step*** $s_w \cdot (1 - r)$ **do**
  for $y \leftarrow 0$ ***to*** H ***step*** $s_h \cdot (1 - r)$ **do**
    Crop patch $P \leftarrow I[x : x + s_w,\ y : y + s_h]$;
    Store patch coordinates (x, y) with P;
    Append $(P, (x, y))$ to S;
  end
end
return S

2.1.2. Detecting sea turtles from high-resolution videos

Computer vision pre-trained models available to the research community are generally trained on image inputs of a fixed size (e.g., 640 × 480) and are then fine-tuned using images of the same size to achieve the optimal performance for the required task. Thanks to transfer learning, a pre-trained foundational model can be applied to a variety of tasks using a smaller dataset than would be required for training from scratch. Furthermore, even if the availability of a high-resolution dataset is not a concern, increasing the resolution of the images increases the model's training requirements in terms of memory and energy consumption. These constraints present challenges for detecting small objects in high-resolution images, particularly in the context of sea turtle detection. For example, cameras capture videos at a resolution of 3840 × 2160 pixels (4K UHD), whereas sea turtles, when recorded at an altitude of 30 m, have an average size of less than 100 × 100 pixels. Consequently, resizing the video frame to a resolution that is acceptable for a pre-trained model would make detecting sea turtles extremely challenging.

To overcome this challenge, our platform uses Slicing Aided Hyper Inference (SAHI) (Akyon et al., 2022), a framework that improves the detection of small objects in high-resolution images. SAHI can operate at two different stages: during model fine-tuning and at inference time. To support improved fine-tuning, a new dataset can be generated by creating overlapping slices from the high-resolution images, which are then used alongside the original images during training. At inference time, SAHI first downscales the high-resolution frame (e.g., from 4K) to the model's fixed input size (e.g., 640 × 480) and performs an initial detection pass. The original high-resolution frame is additionally split into overlapping slices, each resized to the model input dimensions, and inference is performed on each slice. Non-maximum suppression (NMS) (Neubeck and Van Gool, 2006), a technique that eliminates redundant overlapping detections by keeping only the most confident prediction for each turtle, is then applied to merge detections across slices.

While it is possible to train a model to perform inference directly on higher-resolution images for better performance in small object detection (as is the case with high-altitude sea turtle detection), this approach incurs a higher cost in terms of training resources, such as GPU memory and power. Additionally, the higher input resolution would translate to higher GPU requirements at inference time. Conversely, using SAHI reduces training requirements and enables models trained using lower resolution images to be reused.

2.2. Immediate view

Fig. 2 illustrates the Immediate View operating mode with a video file loaded and playing. When using this view, researchers can load a video file and observe the progress of the inference models as the video plays, with the results overlaid on the video canvas. However, the speed at which the user can run inference on the video is limited by its frame rate, making this mode less suitable for long videos. This view comprises four sections:

- *The toolbar* at the top, which grants user access to the session options and enables video and telemetry loading.
- The *video canvas* component, which is responsible for visualizing the loaded video and for drawing detected targets in sync with it. This component is further discussed in Section 2.2.1.
- The *progress bar* component, which is technically part of the video canvas, indicates the playing time of the video.
- The *detections timeline* component: Whenever a target is detected, a marker is placed on the timeline at the corresponding time. While the target remains on screen, the marker grows to indicate how long it was visible for. The marker also allows users to jump to the detection by double-clicking on it, in addition to acting as a visual cue.

If a target species is detected, an entry is added to a detection buffer to collate detection events. If two events for the same detected object occur within one second, they are merged as a single detection event. These collated events are displayed on the timeline component and can be exported at any time to Comma Separated Value (CSV) format, together with additional relevant metadata such as the object's location and size. This allows researchers to analyse the output further using third-party software.

2.2.1. The video canvas component

The video canvas is a custom ReactJS component that enables overlaying detection information such as bounding boxes on video frames. It incorporates a hidden web `<video>` element and a `<canvas>` at its core. The former is used to load and decode the input video, and the latter provides a drawable surface of the same size as the video. Once a video is loaded and the user hits the play button, a request is made to the `<video>` element to provide the image data for the frame at the current execution time. The frame is then processed and the results are composed on the `<canvas>` alongside the original frame data. Although the `<video>` element can display video frames, it is not possible to draw directly on it, so a `<canvas>` is required. Due to processing latency, by the time inference output (bounding boxes and confidence scores) is rendered on the canvas, a new frame may have been displayed in the `<video>` element, potentially resulting in out-of-sync results. Therefore, we decided to hide the `<video>` element and use solely the `<canvas>` to present back to the user.

The video canvas component provides external hooks that allow the application to plug in custom frame processing logic. However, it retains control over the rendering aspects. As the video progresses, the following pipeline is executed for each frame:

1. after a frame becomes ready, it is copied to an `OffscreenCanvas`.
2. raw RGB data is extracted from `OffscreenCanvas` as an `ImageData` object.
3. the `ImageData` is fed to a frame processing callback function. This function calls the active processor, which takes care of performing all the needed transformations to fulfil its job and eventually returns the results of the inference.
4. the raw RGB data is rendered over a temporary `<canvas>` object.

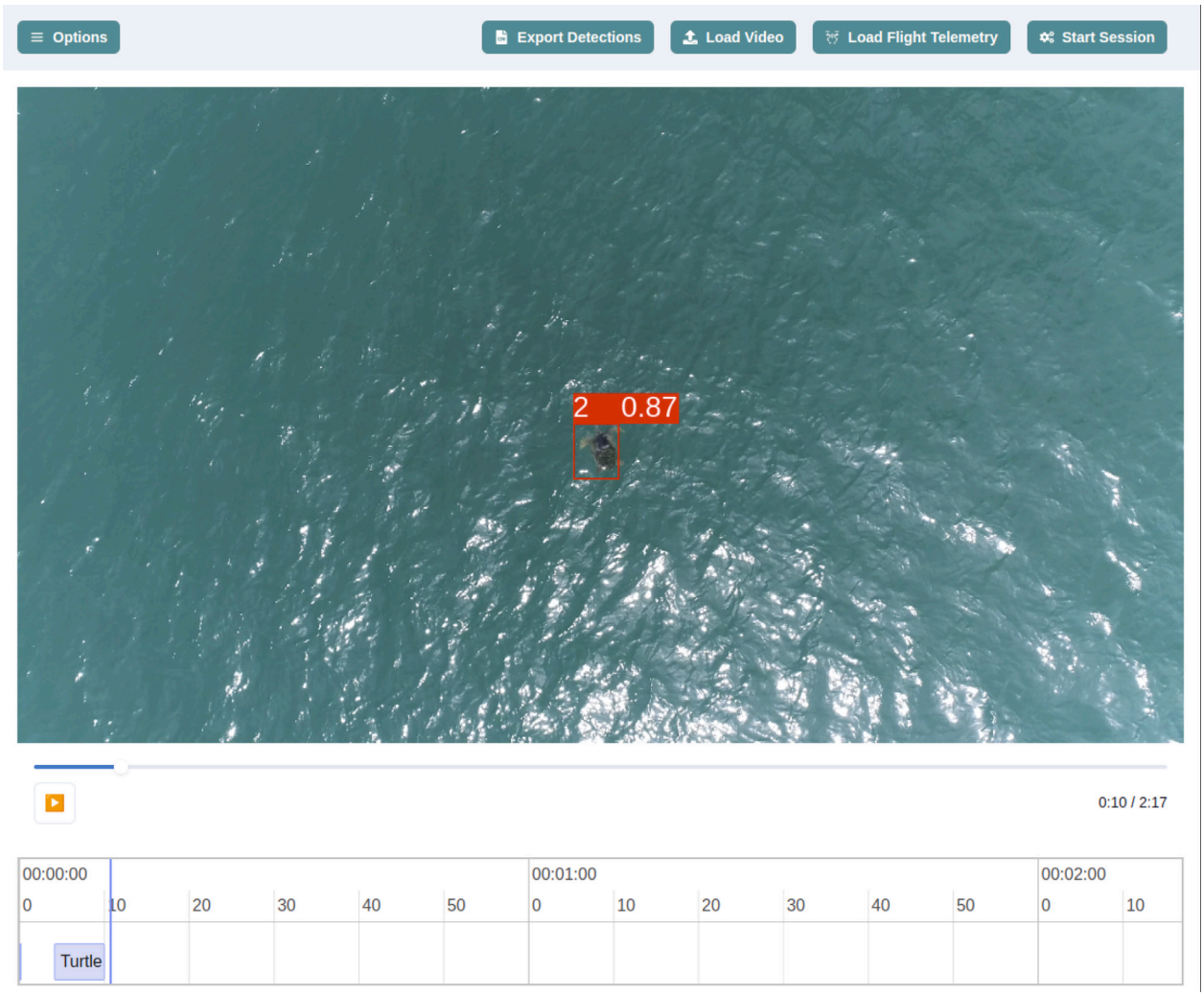


Fig. 2. The immediate view, showing a detection sea turtle being detected in a played video, along with its overlaid bounding box.

5. the mentioned `<canvas>` and the temporary `<canvas>` are delivered to a frame compositing callback function, along with the results from the inference on the analysed frame. Any detection is visually flagged to the user at this step by drawing on the temporary `<canvas>`.
6. the next frame is requested using the `requestVideoFrameCallback` Web API, and the pipeline is executed again.

2.3. Batched view

The batched view (see Fig. 3) enables users to process videos potentially faster than their playing frame-rate, meaning that the detection process is bound to the available hardware resources and model architecture rather than the frame rate of the video. This view provides the same capabilities and features as the *Immediate* view described earlier. However, it does not offer a real-time preview of the detected objects, and it uses fundamentally different technology to power the underlying inference pipeline.

The view is based on the Web Codecs API (MDN, 2024), which is offered on the Web Platform, and is available in the latest browsers. Once a user has selected a video, along with an optional flight telemetry file, the selected files are passed to a Web Worker (MDN, 2025b), which runs further processing outside of the main thread. This prevents the user interface from freezing and degrading the user experience. A processing pipeline is defined and instantiated within the Worker using a transform stream (MDN, 2025a), with each block in the pipeline streaming its output to the next element in the processing chain. The first step in this pipeline reads the raw video file from disk and feeds file chunks to an MP4 demuxer.[4] As we do not want processing to be bound to the playing frame rate, it is not possible to use the video element, which performs demuxing automatically. This means we have to take care of these implementation details manually. As demuxed samples become available, they are processed by the next step in the pipeline, a Video Decoder. Eventually, the decoded frames reach their final destination, which is a custom `WriteableStream` that calls the active processor to run inference on the video frames.

3. Datasets

Data were collected in situ using different UAVs in a variety of light conditions, at different times of day, and in different locations, to record sightings of target marine species, such as sea turtles. The data includes significant variance in sea conditions, hue, sunlight, weather conditions, and seabed complexity (Fig. 4).

The raw videos can be grouped into two categories, based on the altitude of the drone at the time of the recording.

[4] An MP4 demuxer identifies the streams (e.g., video, audio, ...) in the raw stream and makes them available for further processing as individual streams. Our web application uses MP4Box.js (gpac, 2025).

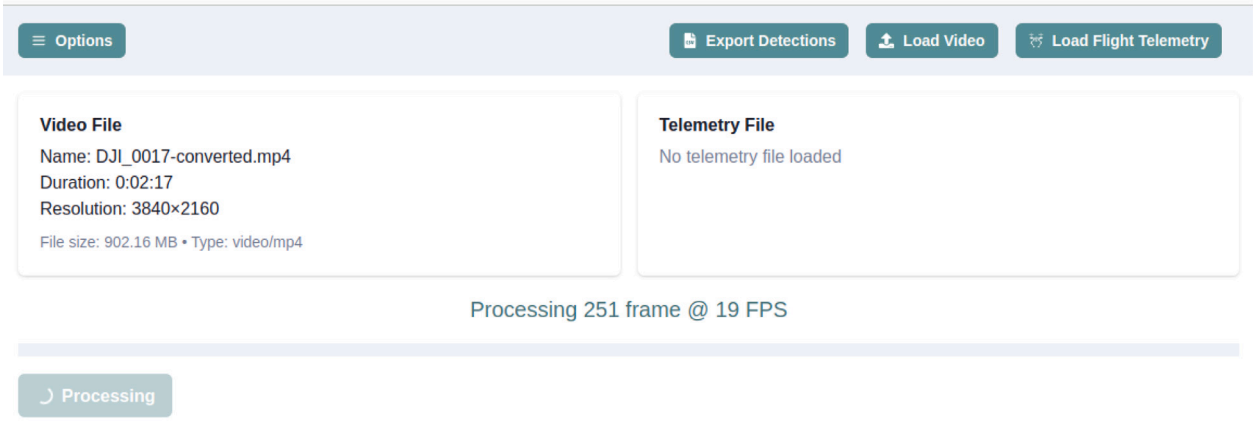
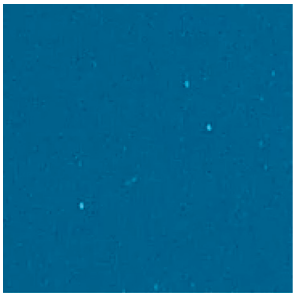


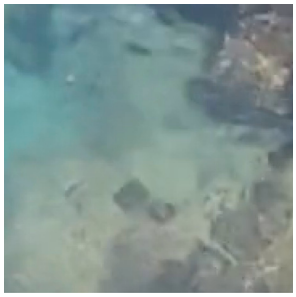
Fig. 3. The batched view with a running inference session.

(a) Sun reflection (glint) on waves

(b) Rocks on the seabed and sun reflection

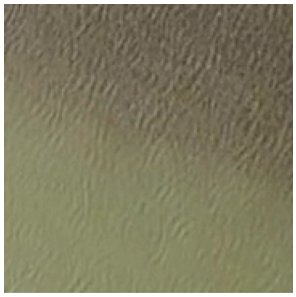


(c) Foam and waves from high altitude (dark blue hue)



(d) Rocks on the seabed from low altitude

(e) Complex seabed from high altitude



(f) Seabed with rocks and clear sand from high altitude on a moving sea

Fig. 4. Representative sample crops of the raw high-resolution images included in the datasets to showcase different challenges to detection in a marine environment.

The first category includes recordings from exploratory UAV flights at mixed altitudes. These videos were recorded using a "DJI Phantom 4 Pro V2" UAV with a camera featuring an aperture range of $F2.8$ to $F11$ and a 24 mm equivalent focal length. Because of the variation in the altitude, the size in pixels of the target species in pixels ranges from tens of pixels at high altitude to hundreds of pixels at low altitude. The distance from the coast also varies, further increasing the detection challenges commonly found in marine environments by introducing complex, non-uniform coastal backgrounds (e.g., partially submerged rocks).

The second category includes videos recorded with a "DJI Phantom 4 Pro" UAV, featuring a camera with an aperture range of $F2.8$ to $F11$. Four videos were recorded in the Amvrakikos Gulf (N 39° 0′ E 21° 0′), Greece, which is a sea turtle foraging site (Rees et al., 2013). Thirty more videos were recorded in Kyparissia Bay (N 37° 21.0′ E 21° 41.5′), Greece, which is a sea turtle nesting area (Margaritoulis et al., 2025). These were recorded at a constant altitude of 40 m and a flight speed of 35 km/h (~9.7 m/s), with the camera at nadir, to survey three 2 km parallel transects approximately 150 m, 300 m and 450 m from shore. The UAV was able to cover each route using a single battery. Speed was selected to give objects approximately 3 s in the visual field without extending the flight time to require two batteries. Altitude was selected to roughly generate a 50 m visual strip-width while maintaining turtles as relatively large and discernible objects in the visual field. 4K video resolution was selected as the maximum available on the drone, which enabled observers to "zoom in" on objects within a frame to more closely determine whether it is a turtle, or potentially to differentiate between individual turtles. Routes were flown when the sea state was between 0 and 2, with a nearly cloud-free sky and at times of day to minimize sun glint.

In these flight conditions, the target species, such as sea turtles, fit within an average bounding box of 70 × 70 pixels. The small size of the detection target within the high-resolution image poses an additional challenge for computer vision models, in addition to those commonly encountered in marine environments (e.g., sun reflection and waves). The raw material includes footage of complex seabeds without any target species, which helps to improve model learning and inference generalization. The raw video material was annotated using the open-source, web-based Computer Vision Annotation Tool (CVAT) (CVAT.ai Corporation, 2023). A sea turtle expert provided temporal and spatial guidance indicating when and where turtles appeared in videos. Three non-experts then generated frame-by-frame bounding box annotations based on this guidance, which were subsequently validated by the experts. Each non-expert annotator fully annotated an entire video. This approach minimized expert time while ensuring annotation accuracy.

Two distinct object detection datasets were created from the two categories mentioned earlier: *Mixed Altitude* and *High Altitude*. Rather

(a) A mixed altitude dataset sample, with a sea turtle on the left on a clear seabed and a mixture of rocks and sand on the right.

(b) A high altitude dataset sample, with a sea turtle just underneath the sea surface on the right.

Fig. 5. Example images from the *mixed altitude* (top) and *high altitude* (bottom).

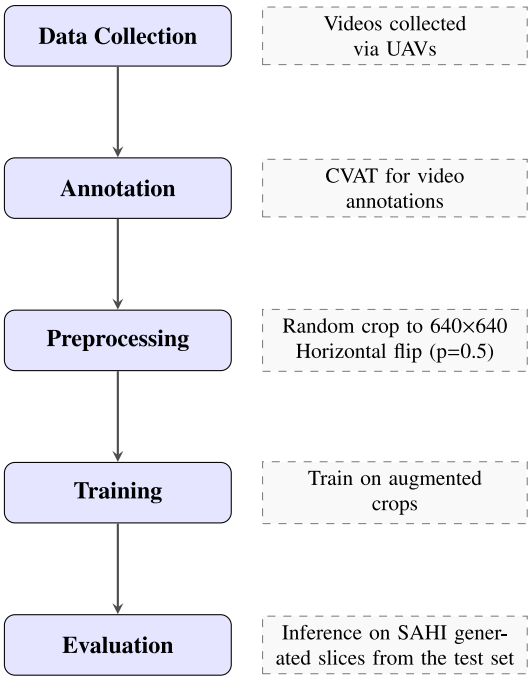


Fig. 6. Training pipeline overview. Input videos are collected via UAVs, annotated using CVAT then preprocessed before being used for training. The evaluation happens on a dedicated test set.

than exporting individual frames from the videos and randomly assigning them to sub-datasets, the full videos were assigned to either a training or a testing set to mitigate information leakage during the training phase (Figueiredo and Mendes, 2024). The training set accounted for 80% of the samples; the frames in the testing set were further divided into a validation set (10%) and a test set (10%).

Finally, the annotations for each dataset were exported in the COCO format (Lin et al., 2014) with frames saved as PNG images. The *Mixed Altitude* dataset comprises 25 442 images with a resolution of 3840 × 2160 pixels. Each image takes up approximately 10 MB of disk space, totalling 178 GB (see Fig. 5(a)). The *High Altitude* dataset contains 6552 images with a resolution of 4096 × 2160, $GSD_w = 1.46$ and $GSD_h = 1.68$, with a total size of 39.1 GB (see Fig. 5(b)).

4. Deep learning models

While building the web-based platform, a variety of different model architectures were tested, with a focus on the most recent and promising real-time object detection model architectures that could provide a sustainable user experience within the constrained browser environment. The training pipelines were prototyped in Python (version 3.11) using the PyTorch (version 2.5.1) machine learning framework with CUDA support, enabling quick experimentation with state-of-the-art techniques.

In order to identify the most effective architectures, we evaluated them based on the following metrics:

- **Mean Average Precision (mAP)**: measures detection accuracy by evaluating how well the model correctly identifies objects and localizes their positions. It is the average of the Average Precision (AP) scores across all classes and Intersection over Union (IoU) thresholds.
- **Inference time**: the time it takes to run inference on a single frame, after pre-processing the input to match the model needs, measured from within the selected processor. This is measured using the ONNX version of the model, using the Python back-end with the CUDA ONNX execution provider, on the local testing machine specified in Section 5.1.
- **ONNX compatibility**: different architectures can use different operators with different maturity, depending on the ONNX provider being used. For example, all operators might be implemented in the WASM provider, but some of them might not be present or be buggy in the newer WebGPU provider.

The development and dry-run of the training pipeline were performed on the same machine used for inference and testing, with the specifications listed in the next section. The final models were produced using distributed training, for which CINECA granted access to the Leonardo cluster (BOOSTER partition) (Turisini et al., 2023). Two nodes, each driven by a single 32-core Intel Ice Lake CPU, were used; each node had four NVidia A100 SXM6 64 GB GPUs.

Transfer learning was employed by fine-tuning the models for 120 epochs, with a learning rate of 10^{-4} using the `AdamW` optimizer (Loshchilov and Hutter, 2019), starting from publicly available model checkpoints pre-trained on the Microsoft COCO dataset (Lin et al., 2014). Light augmentation was applied to the training pipeline. First, each 4K input image was randomly cropped to a 640 × 640 window, enabling the pipeline to see both crops with target classes and background-only crops. Then, with a probability of 0.5, the crop was horizontally flipped. The output of these transformations was then fed to the rest of the training pipeline. The testing set was created by slicing the frames into 640 × 640 crops, with an overlap of 20% across the crops (see Fig. 6 for the full pipeline).

After training, the models are converted to ONNX format, enabling them to be loaded either within a browser-based processor using *onnxruntime-web* or in a local back-end via *onnxruntime-gpu*. This process ensures that the models' inputs are dynamic and uses a target

Table 1
Model performance after fine-tuning on the high altitude dataset. The inference time is measured on a single input, without SAHI.

Model	mAP	Inference time (ms)	Parameters (M)
DeformableDETR (Zhu et al., 2020)	0.47	50	40
RT-DETR (Zhao et al., 2024)	0.50	8	20
D-FINE (Peng et al., 2024)	0.46	7	19

Table 2
Hyperparameters and configuration settings for model training and inference.

Parameter	Value
Training epochs	120
Learning rate	10^{-4}
Optimizer	AdamW
Training input size	640 × 640
Augmentation	Random crop, horizontal flip (p = 0.5)
SAHI slice size	640 × 640 pixels
SAHI overlap ratio	20%
SAHI confidence threshold	0.8
SAHI NMS IoU threshold	0.8

ONNX opset of 16, as well as performing minor optimizations such as constant folding.

The models that have been trained and tested, as reported in Table 1, were selected because they provide state-of-the-art accuracy while keeping inference times and parameter counts low, a combination particularly suitable for the browser computational constraints. The choice was therefore driven by technical considerations rather than ecological ones. DeformableDETR (Zhu et al., 2020) is an object detection architecture featuring multi-scale deformable attention modules, an efficient attention mechanism for processing image feature maps. RT-DETR (Zhao et al., 2024) improves upon DETR architectures by incorporating an efficient hybrid encoder to process multi-scale features, and an uncertainty-minimal query selection to enhance the quality of initial object queries, enabling real-time detections. D-FINE (Peng et al., 2024) redefines the task of predicting target bounding boxes' fixed coordinates as an iterative refinement of probability distributions, significantly enhancing localization accuracy. Although the RT-DETR model's inference time is slightly longer, it has a higher mAP when tested on the high-altitude dataset, making it more suitable for our evaluation.

4.1. Using SAHI

As discussed in Section 2.1.2, to increase the model's ability to detect turtle sightings in high-altitude images, the high-resolution images must be sliced and each slice fed to the model. The original SAHI implementation is written in Python, but most of the pre-processing and post-processing operations are performed using the `numpy` library. While this is convenient and generally not a problem, these constraints prevent potential optimizations as they require data to be moved between the GPU and CPU and prevent pre-processing operations from being converted to ONNX. Therefore, as part of this work, all the SAHI-relevant operations were converted to `torch` to enable SAHI to be used as a component of this application within the browser and via the streamed inference approach.

Additionally, the SAHI slicing configuration was chosen through preliminary tuning to match the characteristics of our dataset and models: slices of 640 × 640 pixels with a 20% overlap and post-processing thresholds of 0.8 for confidence and 0.8 for NMS IoU. All hyperparameters and configuration settings are detailed in Table 2.

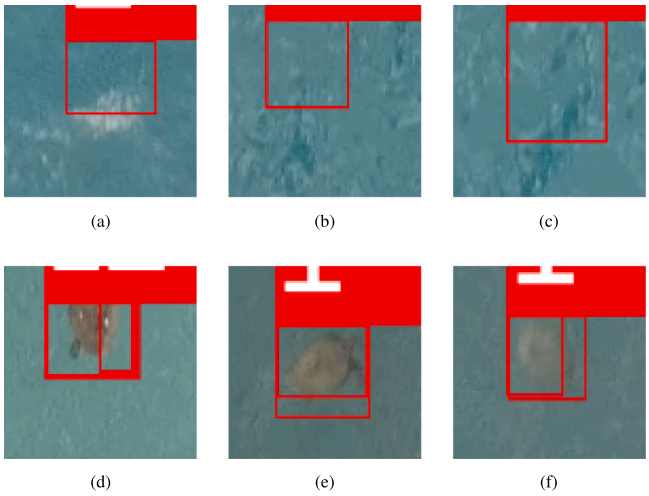


Fig. 7. Images 7(a), 7(b), 7(c) represent false positives due to different sea conditions. 7(d), 7(e), 7(f) represent double counting of an individual turtle attributed to SAHI edge cases.

4.2. Further optional filtering

In our object detection pipeline, which we applied to individual video frames, we observed three main categories of problems (see Fig. 7). The first category comprises false positives caused by challenging sea conditions, such as waves, which can resemble target objects and only last for a few frames. Although improving the detection model could potentially address this issue, we found that a simple temporal filter that removes detections that do not persist for at least three consecutive frames is equally effective. Therefore, we defer model improvements to future work. The second category includes false positives caused by marine litter being misclassified as sea turtles. The third category of problems arises from the use of SAHI (see Section 2.1.2), particularly when detections occur at the edges of image slices and generate overlapping bounding boxes that are not successfully merged by non-maximum suppression. Addressing this issue will likely require modifications to SAHI's post-processing logic, which we also leave for future investigation.

5. Evaluation of the proof of concept

We designed an experiment to evaluate the developed platform (hereinafter referred as PoC). This involved identifying and marking the time reference of each turtle sighting when scanning a series of evaluation videos, i.e., the time a turtle first becomes visible in a video frame and the time it leaves the frame. A panel of three sea turtle experts took part in the experiment, each filling in their findings individually in a spreadsheet designed for this purpose. In addition to noting the time of each sighting, each participant was asked to classify each sighting as either *maybe turtle* or *turtle*. The former captures uncertainty about a specific sighting (e.g., mistaking a rock for a turtle at high altitude), while the latter expresses high confidence. Each expert was also asked to record how long it took them to review each file and perform their analysis. It is important to note that sightings marked as *maybe turtle* are treated as true sightings if at least two experts detected them.

The evaluation videos were also loaded into the Batched View (Section 2.3) to produce machine-generated annotations, which were then compared with the spreadsheets completed by the participants. The videos were processed at their original resolution with different frame-skip settings to evaluate the processing times (Table 3). The final reported results were computed with an 8-frame skip, evaluating three frames for each second of footage shot at 23.967 frames-per-second (FPS).

Table 3
The time it takes to process a 4K video with a duration of 5:28 min, depending on the frame-skip settings.

Frame-skip	Analysis time (mm:ss)
0 (all processed)	50:03
2	26:10
4	13:08
8	6:53
16	3:40

The session was further set to use a streamed processor with the best model, which was trained using the high-altitude dataset (see Table 1) and served from a local Python back-end. It is worth noting that the model could not be used directly in the browser due to a bug (Lochner, 2025) in the `onnxruntime-web` library, which had not yet been fixed at the time of writing. However, once resolved, it will be possible to use the same model without the streamed back-end, directly within the browser, without affecting the accuracy or inference time. Videos were loaded one at a time into the web interface, with the next video only loading after the results of the previous one had been exported to a CSV file. The model was set to report detections with a confidence score > 0.8.

The evaluation material consists of six MP4 videos (h264 compression), with a resolution of 4096 × 2160 pixels at 23.967 FPS, each with a duration of 5 min and 28 s. These videos were recorded using a "DJI Phantom 4 Pro" UAV flying off the Kyparissia Bay (N 37° 21.0′ E 21° 41.5′), Greece. The videos used for this evaluation phase were not included in any of the datasets used for model training or evaluation, and had not been seen by the experts beforehand. These high-altitude videos were selected for the final evaluation as they represent typical operational scenarios for sea turtles monitoring and provide a stringent test of the system's detection capabilities under real-world conditions. While the sample size is limited for a comprehensive ecological validation, it is appropriate to demonstrate the platform's technical feasibility and operational functionality under field conditions.

5.1. Machine specifications

The specifications of the machine used for evaluating the proof-of-concept web application are as follows:

- Processor: AMD Ryzen 9 3950X
- Memory: 128 GB DDR4
- Storage: 1 TB NVMe SSD
- Graphics: NVIDIA GeForce RTX 4090
- Operating System: Ubuntu 24.04

This is a high-performance build capable of handling inference on 4K video streams. Although the application was tested using the Firefox browser (specifically Firefox Nightly 140.0a1, build id 20250522213939), it is designed to be fully compatible with any modern web browser. It can operate seamlessly across different operating systems, except when using a streamed processor. In that case, the host operating system requires a working Python 3.11 environment with `onnxruntime-gpu` support to run the local back-end service.

5.2. Results

The results produced by the participants in the experiments, including the web application, were evaluated using the following approach.

5.2.1. Matching criteria

Let A be the time of first sighting of a turtle and B the time of the last visible position of the turtle in a video: together, they identify the detected sighting window $[A, B]$. Let G_A and G_B be the points in time identifying the ground truth time window for a sighting $[G_A, G_B]$. We define two matching criteria:

- **Strict matching**. A detected sighting $[A, B]$ matches a ground truth sighting $[G_A, G_B]$ if $A \geq G_A$ and $B \leq G_B$, meaning that the detected window is fully contained within the ground truth window.
- **Relaxed matching**. A detected sighting $[A, B]$ matches a ground truth sighting $[G_A, G_B]$ if $A \geq G_A - \Delta t$ and $B \leq G_B + \Delta t$. This criterion accounts for minor temporal imprecision in the detected time windows by extending the ground truth window by Δt on both sides. We set $\Delta t = 3$ s as this represents the typical temporal error observed in expert annotations, where most experts correctly identified sighting windows but were off by at most 3 s on each end.

Each detected sighting is matched to the closest ground truth sighting (if any) that satisfies the chosen criterion.

5.2.2. Evaluation metrics

Using the matching criteria defined above, we compute the following metrics:

- **Precision**. The percentage of detected sightings that match a ground truth sighting. This measures the proportion of detections that are correct.
- **Recall**. The percentage of ground truth sightings that are matched by at least one detected sighting. This measures the proportion of actual turtle sightings that were successfully detected.
- **Analysis time**. The time that a participant used to analyse a given video and annotate the sightings, including the time taken to consult with other experts, if needed.

Both precision and recall are computed using strict and relaxed matching criteria, yielding four detection quality metrics in total.

The results of the evaluation are sketched in Table 4. In videos `DJI_0001`, `DJI_0002`, and `DJI_0003`, the precision and recall of the PoC are comparable to those of the domain experts. For these videos, the analysis time of the system is similar to that taken by the other participants. The initial videos are simpler, featuring good sea conditions and easily identifiable turtles. The last three videos (`DJI_0004`, `DJI_0028`, `DJI_0029`) present increased complexity, which is reflected in the domain experts' longer analysis time, while the PoC analysis time remains constant. This increased complexity makes it moderately more challenging for human experts to achieve perfect precision and recall. This results in misaligned detection windows and occasional identification errors, as highlighted by the fact that relaxed precision and recall are consistently higher than their strict counterparts for several experts. This is to be expected, given that domain experts have to manually pause the video and record the time. The last two videos are significantly more complex, featuring multiple turtles per frame and more challenging sea conditions. In these cases, the PoC exhibits substantially lower precision (0.33–0.44) than domain experts, due to a high false positive rate, though it maintains perfect recall (1.00) in `DJI_0029` and reasonable recall (0.67) in `DJI_0028`. When naive filtering is applied, as discussed in Section 4.2, the filtered PoC's precision improves dramatically to 0.82–0.92, becoming comparable to or exceeding that of the domain experts, while maintaining or even improving recall (0.93–1.00).

Although the potential for improvement may seem limited, the main benefit is that it reduces the need for domain experts to be present throughout the video processing stage. Their expertise can instead be focused on reviewing and validating the results afterwards. In this

Table 4
Proof of concept validation results: how the model and experts perform in terms of the accuracy and analysis time of sightings. The *Turtle Count* column refers to the number of turtles identified by the participant, including false positives.

Video	Participant	Precision	Recall	Relaxed precision	Relaxed recall	Turtle count	Ground truth	Analysis time (s)
DJI_0001	Expert 1	1.00	1.00	1.00	1.00	2	2	420
	Expert 2	1.00	1.00	1.00	1.00	2	2	600
	Expert 3	1.00	1.00	1.00	1.00	2	2	523
	PoC	1.00	1.00	1.00	1.00	2	2	413
DJI_0002	Expert 1	0.67	0.67	1.00	1.00	3	3	660
	Expert 2	1.00	1.00	1.00	1.00	3	3	600
	Expert 3	1.00	1.00	1.00	1.00	3	3	609
	PoC	0.67	0.67	1.00	1.00	3	3	413
DJI_0003	Expert 1	1.00	1.00	1.00	1.00	0	0	360
	Expert 2	1.00	1.00	1.00	1.00	0	0	300
	Expert 3	1.00	1.00	1.00	1.00	0	0	331
	PoC	1.00	1.00	1.00	1.00	0	0	413
DJI_0004	Expert 1	1.00	1.00	1.00	1.00	5	5	660
	Expert 2	0.33	0.40	0.83	1.00	6	5	900
	Expert 3	1.00	1.00	1.00	1.00	5	5	718
	PoC	0.80	0.80	0.80	0.80	4	5	413
DJI_0028	Expert 1	1.00	0.67	1.00	0.80	12	15	720
	Expert 2	0.94	0.67	0.94	0.87	15	15	1080
	Expert 3	1.00	1.00	1.00	1.00	15	15	948
	PoC	0.33	0.67	0.39	0.56	20	15	413
	PoC filtered[a]	0.82[a]	0.93[a]	0.82[a]	0.93[a]	18[a]	15	413
DJI_0029	Expert 1	1.00	1.00	1.00	1.00	44	44	1200
	Expert 2	1.00	0.73	1.00	0.89	39	44	1500
	Expert 3	1.00	0.84	1.00	0.98	43	44	943
	PoC	0.44	1.00	0.56	1.00	78	44	413
	PoC filtered[a]	0.92[a]	1.00[a]	0.92[a]	1.00[a]	48[a]	44	413

[a] PoC results obtained via the naive filter described in Section 4.2.

context, the domain experts emphasized that it is better to have a higher number of false positives than to miss actual turtle detections, since the former can be dismissed quickly, whereas the latter represents a missed conservation opportunity.

5.3. Limitations of the proposed approach

The use of the PoC revealed some potential limitations, which will be addressed as the technology matures and as part of future work for the platform:

- **WebGPU browser maturity**. Different web browsers have different maturity levels of their WebGPU implementation and, due to this, the quality of the experience may vary across browser vendors and versions. The WebGPU implementation status across browsers is documented by the GPU for the web community group (W3C GPU for the Web Community Group, 2025). The PoC was tested with Firefox Nightly 140.0a1 (build ID 202505222 13939) and requires a dedicated GPU for in-browser inference.
- **Inconsistent performance**. When using in-browser processing, inference time heavily depends on the specifications of the machine being used and might differ from the results outlined in the previous sections. In contrast, cloud-based processing provides more consistent performance.
- **High false-positives rate in complex videos**. The output of the detection model is currently filtered by the simple mechanism described in Section 4.2. We believe that using object tracking algorithms will address this limitation.
- **Video file size limitations**. The maximum file size of a video file that can be processed by the web application is limited by the amount of RAM on the user machine and any browser-dependent restrictions. The video files used for our evaluation ranged from 1.2 GB to 4.1 GB. We empirically found an optimal video file size to be around 2 GB.
- **SRT flight telemetry limitations**. While easier to access and process, the DJI SRT-based flight telemetry is recorded at 1 Hz, affecting synchronization accuracy, and lacks some of the data points available in the more comprehensive DAT/TXT flight logs (e.g. orientation).
- **Limited ecological generalizability**. The models trained for our study are based on a limited dataset. Broader ecological validation of these models requires larger and more diverse training datasets.

6. Conclusions and future work

This paper details the development of a local-first web application designed to automatically assess sea turtle distribution and speed up the disambiguation process during manual reviews. We successfully collected, annotated, and reviewed extensive footage to create two distinct prototype datasets: one comprising frames from mixed altitudes, and one comprising frames taken at a standard height. State-of-the-art models were then trained on these datasets. The best-performing models were exported to ONNX and seamlessly integrated into the web application. This enables researchers to process footage locally, leveraging their computational resources and eliminating the need for third-party cloud services. This work lays a solid foundation for future enhancements, including local model loading, post-processing refinement, replaying detection files over videos, generating statistical summaries, deployment on a public website.

Building on this foundational platform, we plan to conduct extensive ecological validation in our upcoming data collection campaigns. This will include rigorous quantitative assessments demonstrating how the tool improves population estimates, habitat use analysis and behavioural interpretation. Such comprehensive validation will leverage the platform's standardized approach to establish robust ecological benchmarks.

Furthermore, we intend to make additional pre-trained models available in the web application and provide users with guidance on which model to use, depending on their task. An intriguing aspect we plan to explore is enabling users to fine-tune models on their own datasets, an untapped possibility enabled by the latest technologies. Empowering experts to perform annotations within the browser and use their local hardware to fine-tune existing models would significantly enhance the platform's flexibility and adaptability to diverse

research contexts. As we continue to collect videos through ad hoc UAV campaigns, we will perform a more rigorous comparison of model architectures. These next steps will further enhance the application's usability and impact in the field of sea turtle monitoring and conservation. In addition to our ongoing commitment to maintenance, we hope that opening the platform's source code and granting a very permissive license (MIT) will seed a community that will contribute updates and new features. The open nature of the platform means that its use is not limited to sea turtle identification, but through scientists uploading their own models it can be used for research on other marine megafauna, such as cetaceans, and even other ecological contexts such as terrestrial ecological surveying.

CRediT authorship contribution statement

Alessio Pierluigi Placitelli: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Data curation, Conceptualization. **Sandra Hochscheid:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Data curation, Conceptualization. **Fulvio Maffucci:** Writing – review & editing, Writing – original draft, Methodology, Investigation, Data curation, Conceptualization. **ALan F. Rees:** Writing – review & editing, Data curation, Conceptualization. **Gianluca Treglia:** Writing – review & editing, Investigation, Data curation. **Antonino Staiano:** Writing – review & editing, Writing – original draft, Supervision, Methodology, Conceptualization.

Acknowledgement

We acknowledge ISCRA for awarding this project access to the LEONARDO supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CINECA (Italy). Video data collection in Greece was funded under the European Union LIFE Euroturtles project (LIFE15 NAT/HR/000997).

Data availability

Code is available on Github as linked in the paper (along with trained models). Source images cannot be shared due to NDA constraints at this time.

References

Agabiti, C., Tolve, L., Baldi, G., Zucchini, M., Tuccio, S., Restelli, F., Freggi, D., Luschi, P., Casale, P., 2024. Combining UAVs and multi-sensor dataloggers to estimate fine-scale sea turtle density at foraging areas: a case study in the central Mediterranean. Endanger. Species Res. 54, 395–408.

Akyon, F.C., Altinuc, S.O., Temizel, A., 2022. Slicing aided hyper inference and fine-tuning for small object detection. In: 2022 IEEE International Conference on Image Processing. ICIP, IEEE, pp. 966–970.

Axford, D., Sohel, F., Vanderklift, M.A., Hodgson, A.J., 2024. Collectively advancing deep learning for animal detection in drone imagery: Successes, challenges, and research gaps. Ecol. Inform. 83, 102842. http://dx.doi.org/10.1016/j.ecoinf.2024.102842, URL: https://www.sciencedirect.com/science/article/pii/S1574954124003844.

Berger-Wolf, T.Y., Rubenstein, D.I., Stewart, C.V., Holmberg, J.A., Parham, J., Menon, S., Crall, J., Van Oast, J., Kiciman, E., Joppa, L., 2017. Wildbook: Crowdsourcing, computer vision, and data science for conservation. arXiv preprint arXiv:1710.08880.

Bevan, E., Whiting, S., Tucker, T., Guinea, M., Raith, A., Douglas, R., 2018. Measuring behavioral responses of sea turtles, saltwater crocodiles, and crested terns to drone disturbance to define ethical operating thresholds. PLoS One 13 (3), e0194460.

CVAT.ai Corporation, 2023. Computer Vision Annotation Tool (CVAT). URL: https://github.com/cvat-ai/cvat.

Dickson, L.C., Negus, S.R., Eizaguirre, C., Katselidis, K.A., Schofield, G., 2022. Aerial drone surveys reveal the efficacy of a protected area network for marine megafauna and the value of sea turtles as umbrella species. Drones 6 (10), 291.

Dimauro, G., Simone, L., Carlucci, R., Fanizza, C., Lomonte, N., Maglietta, R., 2022. Automated and non-invasive UAV-based system for the monitoring and the group size estimation of dolphins. In: 2022 7th International Conference on Smart and Sustainable Technologies. SpliTech, IEEE, pp. 1–8.

Dujon, A.M., Ierodiaconou, D., Geeson, J.J., Arnould, J.P., Allan, B.M., Katselidis, K.A., Schofield, G., 2021. Machine learning to detect marine animals in UAV imagery: effect of morphology, spacing, behaviour and habitat. Remote. Sens. Ecol. Conserv. 7 (3), 341–354.

Duporge, I., Spiegel, M.P., Thomson, E.R., Chapman, T., Lamberth, C., Pond, C., Macdonald, D.W., Wang, T., Klinck, H., 2021. Determination of optimal flight altitude to minimise acoustic drone disturbance to wildlife using species audiograms. Methods Ecol. Evol. 12 (11), 2196–2207. http://dx.doi.org/10.1111/2041-210X.13691, arXiv:https://besjournals.onlinelibrary.wiley.com/doi/pdf/10.1111/2041-210X.13691, URL: https://besjournals.onlinelibrary.wiley.com/doi/abs/10.1111/2041-210X.13691.

Figueiredo, R.B., Mendes, H.A., 2024. Analyzing information leakage on video object detection datasets by splitting images into clusters with high spatiotemporal correlation. IEEE Access.

Gao, Z., Bird, C., Barr, E.T., 2017. To type or not to type: quantifying detectable bugs in JavaScript. In: 2017 IEEE/ACM 39th International Conference on Software Engineering. ICSE, IEEE, pp. 758–769.

Goh, H.-A., Ho, C.-K., Abas, F.S., 2023. Front-end deep learning web apps development and deployment: a review. Appl. Intell. 53 (12), 15923–15945.

Gonzalez Nunez, J.A., Gonzalez Nunez, J.G., Akbas, M.I., Currier, P., Macchiarella, N.D., 2024. Real-time detection of sea turtles using UAV and neural networks on edge devices. J. Aviat./Aerosp. Educ. Res. 33 (5), 1.

gpac, 2025. Mp4box.js. URL: https://github.com/gpac/mp4box.js/ (Accessed 24 March 2025).

Gray, P.C., Fleishman, A.B., Klein, D.J., McKown, M.W., Bezy, V.S., Lohmann, K.J., Johnston, D.W., 2019. A convolutional neural network for detecting sea turtles in drone imagery. Methods Ecol. Evol. 10 (3), 345–355.

Kleppmann, M., Wiggins, A., Van Hardenberg, P., McGranaghan, M., 2019. Local-first software: you own your data, in spite of the cloud. In: Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software. pp. 154–178.

Lim, Y., Singer, D., 2006. MIME Type Registration for MPEG-4. In: RFC 4337. RFC Editor, http://dx.doi.org/10.17487/RFC4337, URL: https://www.rfc-editor.org/info/rfc4337.

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. In: Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13. Springer, pp. 740–755.

Lochner, J., 2025. Using ceil() in shape computation is not yet supported for AveragePool. URL: https://github.com/microsoft/onnxruntime/issues/21206 (Accessed 01 April 2025).

Loshchilov, I., Hutter, F., 2019. Decoupled weight decay regularization. In: ICLR 2019.

Margaritoulis, D., Rees, A.F., Riggall, T.E., 2025. Exponential increase in a loggerhead sea turtle nesting population: Investigating the role of multi-decadal nest protection in Kyparissia Bay, Greece. Zoöl. Stud. 64.

Martínez-Movilla, A., Rodríguez-Somoza, J.L., Román, M., Olabarria, C., Martínez-Sánchez, J., 2024. Rapid diagnosis of the geospatial distribution of intertidal macroalgae using large-scale UAVs. Ecol. Inform. 83, 102845. http://dx.doi.org/10.1016/j.ecoinf.2024.102845, URL: https://www.sciencedirect.com/science/article/pii/S157495412400387X.

MDN, 2024. WebCodecs API. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebCodecs_API (Accessed 24 March 2025).

MDN, 2025a. TransformStream - Web APIs. URL: https://developer.mozilla.org/en-US/docs/Web/API/TransformStream (Accessed 24 March 2025).

MDN, 2025b. Web workers API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Web_Workers_API (Accessed 24 March 2025).

Neubeck, A., Van Gool, L., 2006. Efficient non-maximum suppression. In: 18th International Conference on Pattern Recognition. ICPR'06, Vol. 3, pp. 850–855. http://dx.doi.org/10.1109/ICPR.2006.479.

Noguchi, N., Nishizawa, H., Shimizu, T., Okuyama, J., Kobayashi, S., Tokuda, K., Tanaka, H., Kondo, S., 2025. Efficient wildlife monitoring: Deep learning-based detection and counting of green turtles in coastal areas. Ecol. Inform. 86, 103009.

ONNX Runtime developers, 2018. ONNX runtime. URL: https://github.com/microsoft/onnxruntime.

Papazekou, M., Kyprioti, A., Chatzimentor, A., Dimitriadis, C., Vallianos, N., Mazaris, A.D., 2024. Advancing sea turtle monitoring at nesting and near shore habitats with UAVs, data loggers, and state of the art technologies. Divers. 16 (3), 153.

Pedrazzi, L., Naik, H., Sandbrook, C., Lurgi, M., Fürtbauer, I., King, A.J., 2025. Advancing animal behaviour research using drone technology. Animal Behav. 222, 123147. http://dx.doi.org/10.1016/j.anbehav.2025.123147, URL: https://www.sciencedirect.com/science/article/pii/S0003347225000740.

Peng, Y., Li, H., Wu, P., Zhang, Y., Sun, X., Wu, F., 2024. D-FINE: Redefine regression task in DETRs as fine-grained distribution refinement. arXiv:2410.13842.

Ramírez, S., 2025. Fullstack FastAPI template. URL: https://github.com/fastapi/full-stack-fastapi-template.git.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You only look once: Unified, real-time object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 779–788.

Rees, A.F., Alfaro-Shigueto, J., Barata, P., Bjorndal, K.A., Bolten, A.B., Bourjea, J., Broderick, A., Campbell, L., Cardona, L., Carreras, C., et al., 2016. Are we working towards global research priorities for management and conservation of sea turtles? Endanger. Species Res. 31, 337–382.

Rees, A.F., Avens, L., Ballorain, K., Bevan, E., Broderick, A.C., Carthy, R.R., Christianen, M.J., Duclos, G., Heithaus, M.R., Johnston, D.W., et al., 2018. The potential of unmanned aerial systems for sea turtle research and conservation: a review and future directions. Endanger. Species Res. 35, 81–100.

Rees, A.F., Margaritoulis, D., Newman, R., Riggall, T.E., Tsaros, P., Zbinden, J.A., Godley, B.J., 2013. Ecology of loggerhead marine turtles Caretta caretta in a neritic foraging habitat: movements, sex ratios and growth rates. Mar. Biol. 160, 519–529.

Robinson, N.J., Bigelow, W.F., Cuffley, J., Gary, M., Hoefer, S., Mills, S., Smith, A., Miguel Blanco, A., 2020. Validating the use of drones for monitoring the abundance and behaviour of juvenile green sea turtles in mangrove creeks in The Bahamas. Testudo 9 (2), 24–35.

Subeesh, A., Prakash Kumar, S., Kumar Chakraborty, S., Upendar, K., Singh Chandel, N., Jat, D., Dubey, K., Modi, R.U., Mazhar Khan, M., 2024. UAV imagery coupled deep learning approach for the development of an adaptive in-house web-based application for yield estimation in citrus orchard. Meas. 234, 114786. http://dx.doi.org/10.1016/j.measurement.2024.114786, URL: https://www.sciencedirect.com/science/article/pii/S0263224124006717.

Sykora-Bodie, S.T., Bezy, V., Johnston, D.W., Newton, E., Lohmann, K.J., 2017. Quantifying nearshore sea turtle densities: applications of unmanned aerial systems for population assessments. Sci. Rep. 7 (1), 17690.

Turisini, M., Amati, G., Cestari, M., 2023. Leonardo: A pan-european pre-exascale supercomputer for HPC and AI applications. arXiv preprint arXiv:2307.16885.

W3C GPU for the Web Community Group, 2025. Implementation status. URL: https://github.com/gpuweb/gpuweb/wiki/Implementation-Status (Accessed 22 May 2025).

Wang, C.-Y., Bochkovskiy, A., Liao, H.-Y.M., 2023. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 7464–7475.

Yaney-Keller, A., San Martin, R., Reina, R.D., 2021. Comparison of UAV and boat surveys for detecting changes in breeding population dynamics of sea turtles. Remote. Sens. 13 (15), 2857.

Zhao, Y., Lv, W., Xu, S., Wei, J., Wang, G., Dang, Q., Liu, Y., Chen, J., 2024. DETRs beat YOLOs on real-time object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. CVPR, pp. 16965–16974.

Zhou, P., Zhou, Z., Yang, C., Fang, Y., Bu, Y.-X., Wang, C.-S., Zhang, D.-S., Shen, H.-B., Pan, X., 2025. Development of a multi-modal deep-learning-based web application for image classification of marine decapods in conservation practice. Front. Mar. Sci. Volume 12 - 2025, http://dx.doi.org/10.3389/fmars.2025.1496831, URL: https://www.frontiersin.org/journals/marine-science/articles/10.3389/fmars.2025.1496831.

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., Dai, J., 2020. Deformable DETR: Deformable transformers for end-to-end object detection. arXiv preprint arXiv:2010.04159.