MANCHESTER 1824
The University of Manchester

A Scalable Synthetic Image Generation Pipeline and Lightweight Deep Learning-based Object Detection Framework for Real-Time Space Debris Detection

Third Year Individual Project - Final Report

Year of submission

2025

Student ID

10828417

Supervisor: Dr. Frank Podd

School of Engineering

Contents

Word count: 12,744

List of figures

List of tables

Abstract

This paper investigates the feasibility of using a synthetic image dataset and a lightweight deep learning model for real-time space debris detection, aiming to support the development of safer and more autonomous Active Debris Removal (ADR) missions. The growing threat of space debris to satellite constellations necessitates prompt mitigation measures. However, progress in debris mitigation technologies has been hindered by the scarcity of real-world images and the limited computational capacity of onboard systems—both of which are bottlenecks for object detection-based approaches. To address these challenges, two modular pipelines were developed: one for synthetic image generation using Blender, and another for real-time debris detection using machine learning tools. The image generation pipeline produced a dataset of 2,400 annotated, photorealistic images across four debris classes, with randomized lighting, poses, and orbital conditions. Post-render augmentation introduced artifacts such as Gaussian and Poisson noise, motion blur, and lens distortion to enhance realism. The detection pipeline trained a YOLOv8n model on this dataset and evaluated its performance using various metrics. The model achieved an mAP@50 of 0.995 and an mAP@[50:95] of 0.9767, outperforming YOLOv11n and YOLOv12n in balancing accuracy and efficiency. Inference on both unseen synthetic and real-world debris images confirmed the model's generalizability, although some positional bias was observed. Further analysis of camera and debris coordinates in 3D space verified the uniform distribution of positions and orientations, demonstrating the visual variability of the dataset. These results show that randomized and augmented synthetic datasets can effectively train object detection models for space debris identification, offering a scalable, low-cost alternative to in-orbit data collection. The compatibility of the pipelines with consumer-grade hardware and their automation capabilities support their potential use in vision-based on-board navigation for ADR missions. Future work could focus on narrowing the synthetic-to-real domain gap and integrating pose estimation to ex[and the practical utility of the proposed approach.

Declaration of originality

I hereby confirm that this dissertation is my own original work unless referenced clearly to the contrary, and that no portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Intellectual property statement

i The author of this thesis (including any appendices and/or schedules to this thesis) owns certain copyright or related rights in it (the "Copyright") and s/he has given The University of Manchester certain rights to use such Copyright, including for administrative purposes.

ii Copies of this thesis, either in full or in extracts and whether in hard or electronic copy, may be made only in accordance with the Copyright, Designs and Patents Act 1988 (as amended) and regulations issued under it or, where appropriate, in accordance with licensing agreements which the University has from time to time. This page must form part of any such copies made.

iii The ownership of certain Copyright, patents, designs, trademarks and other intellectual property (the "Intellectual Property") and any reproductions of copyright works in the thesis, for example graphs and tables ("Reproductions"), which may be described in this thesis, may not be owned by the author and may be owned by third parties. Such Intellectual Property and Reproductions cannot and must not be made available for use without the prior written permission of the owner(s) of the relevant Intellectual Property and/or Reproductions.

iv Further information on the conditions under which disclosure, publication and commercialisation of this thesis, the Copyright and any Intellectual Property and/or Reproductions described in it may take place is available in the University IP Policy (see `http://documents.manchester.ac.uk/DocuInfo.aspx?DocID=24420`), in any relevant Dissertation restriction declarations deposited in the University Library, and The University Library's regulations (see `http://www.library.manchester.ac.uk/about/regulations/_files/Library-regulations.pdf`).

1 Introduction

1.1 Background

Artificial intelligence (AI), a term first coined at the Dartmouth Conference in 1956 [1], broadly refers to the use of technologies to develop machines and computers capable of mimicking cognitive functions associated with human intelligence [2]. The concept has been around for decades, with its origins tracing back to the mid-20th century when Alan Turing introduced the Turing Test, a method to determine whether a machine can exhibit intelligent behavior [1]. Since then, each decade of AI development has built upon the last, reflecting human ingenuity and a continuous drive towards progress. As you read this sentence, AI algorithms are painting cosmic portraits, composing symphonies, predicting protein structures, and even providing personalized financial advice. In short, AI has already had a profound and far-reaching impact on our lives.

As we enter the fifth industrial revolution with AI taking center stage, the space industry, though booming, has remained somewhat in the background. Advanced propulsion systems, reusable launch vehicles, space tourism, exploration, asteroid mining, and real estate are no longer futuristic concepts but emerging realities shaping our future beyond Earth. However, the vastness, harsh conditions, and intense radiation of space contribute to the high costs, increased complexity, and prolonged timescales of space missions. Consequently, these challenges are precisely where AI is proving to be remarkably effective, enabling missions that were once beyond human reach [3].

While AI is transforming industries, it is Machine Learning (ML)—a subset of AI—that is driving many of the most significant advancements [4]. ML algorithms extract knowledge from data, continuously refining their performance through autonomous learning [2]. In the realm of space, ML has become an indispensable tool in this data-driven era, where the volume of information generated by spacecraft and satellites is astronomical. Today, it is reasonable to estimate that all active satellites and spacecraft collectively produce over 100 petabytes (PB) of data each day. To put that number into perspective, that's equivalent to filling 781,000 iPhones (128GB each) to capacity every single day or storing every book ever written 2,000 times over—every single day.

1.2 Motivation

The space industry is advancing toward its next frontier at an unprecedented pace, with major developments emerging across the globe every week, a trend reflected in the sector's exponential economic growth. The global space economy is projected to reach $1.8 trillion by 2035, up from $630 billion in 2023 [5].

However, this rapid expansion has led to increasing congestion in Low-Earth Orbit (LEO), directly contributing to the growing issue of space debris. "Space debris is defined as any anthropogenic space object that is no longer in active use" [6, para. 2]. Defunct satellites and spent rocket bodies account for only a small fraction of the estimated 32,000 objects larger than 10 cm currently in LEO [6]. Yet, these are vastly outnumbered by an estimated 131 million smaller fragments, which pose an even greater

threat due to their high velocity and unpredictable trajectories [2].

With current space debris regulations lacking enforcement mechanisms [7] and the rapid increase in satellite deployments, the debris problem shows no signs of slowing down. If left unchecked, it could create a cascading crisis, given humanity's increasing reliance on space services for everyday activities. More satellites mean more collisions. More collisions generate more debris, increasing the likelihood of further collisions, perpetuating a self-sustaining cycle. This runaway effect, known as the Kessler Syndrome [6], threatens to render certain orbits unusable.

The need to mitigate the space debris threat is undeniable. While new mitigation policies, such as the FCC's 5-year rule for deorbiting satellites [8], have been introduced, they are only enforced within the jurisdictions of the countries that implement them—in this case, the United States. Consequently, additional mitigation efforts are necessary, with Active Debris Removal (ADR) missions emerging as a widely considered solution. These missions deploy a chaser spacecraft (chaser) equipped with technologies such as robotic arms, nets, harpoons, or ion beams to capture debris, which is then either deorbited or relocated to a graveyard orbit [9].

However, ADR missions are inherently complex due to the uncooperative and unpredictable nature of space debris [10]. Defunct satellites and fragments lack the ability to communicate their position, orientation velocity, or condition, leaving the chaser to operate with limited information. Many of these objects have sustained physical damage from prolonged exposure to space, further complicating capture efforts.

In addition, the high costs and long timescales of space missions make preliminary inspection trips impractical. Ground-based surveillance networks provide some insight, but they can only track larger objects, leaving millions of smaller, yet equally hazardous, fragments unmonitored. As the density of debris increases, reliance on these systems will become unsustainable.

For ADR missions to be effective, the chaser must accurately determine the target debris' position and orientation (pose) relative to itself. Without this information, planning and executing maneuvers for tracking, proximity operations, and capture becomes significantly more challenging. Pose estimation in orbit is facilitated by sensors attached to the chaser, which observe the debris. For example, the RemoveDEBRIS mission, led by the Surrey Space Center, used a Vision-Based Navigation (VBN) system consisting of a 2-dimensional (2D) camera and a 3-dimensional (3D) Light Detection and Ranging (LiDAR) device [11]. Similarly, ELSA-d, a mission developed and operated by Astroscale, used optical sensors on the chaser to detect and track the client satellite, which also simulated debris [12].

To estimate the pose of debris, the outputs of these sensors have traditionally been processed using feature extractors such as the Scale-Invariant Feature Transform (SIFT), a patented classical computer vision algorithm [10]. However, while effective in some cases, SIFT is limited by its computational cost, adaptability, and robustness to harsh environmental conditions. Recent advancements in ML have introduced deep learning (DL)-based object detection models, which offer superior performance but present two major challenges:

1. **Data Requirements:** DL models require large, diverse datasets for training; without sufficient data,

they struggle to generalize and may produce inaccurate predictions.

2. **Computational Constraints:** They rely on powerful Graphics Processing Units (GPUs) for training and inference, making deployment on the chaser challenging due to strict power, weight, and computational limitations.

Consequently, this project aims to address both challenges by exploring synthetic image datasets and lightweight object detection models, and lays the foundation to make DL-based pose estimation feasible for space-based applications.

1.3 Aims and objectives

The aim of this project is to contribute to the development of technological solutions addressing the growing threat of space debris—comprising defunct satellites, non-functional spacecraft, rocket bodies, and fragments—impacting current space operations and future expeditions. To support and accelerate ongoing space debris mitigation efforts, particularly ADR missions, this project will develop two accessible, efficient, and scalable pipelines for synthetic image generation and real-time space debris detection. The pipeline will be implemented using a diverse software suite, assessed for their suitability in the intended application, and benchmarked against recent solutions in the field. The main objectives of this project are summarized below:

- Design a synthetic image generation pipeline using Blender to produce a high-resolution, photorealistic, and diverse space debris images for model training.
- Implement domain randomization and data augmentation techniques to enhance synthetic image dataset variability and improve model generalization to real-world conditions.
- Develop a lightweight DL framework using YOLOv8n for real-time object detection of uncooperative and unpredictable space debris.
- Train the object detection model on the synthetic image dataset, and use metrics such as mAP@50 and mAP@[50:95] to evaluate its performance.
- Benchmark the proposed pipeline and trained models against recent YOLO variants (YOLOv11n and YOLOv12n) to assess scalability, accuracy, and computational efficiency.
- Test the best-performing model on a limited set of real-world debris images to evaluate its ability to generalize beyond synthetic data.
- Visualize and analyze spatial metadata (pose) generated during rendering to highlight rotational and positional variation in the synthetic image dataset.

1.4 Report structure

This report is organized as follows. Section 2 reviews the existing literature, highlighting relevant strengths and limitations, and identifying key research gaps within the topic. Section 3 outlines the

methodology used to develop the synthetic image generation and space debris detection pipelines, addressing what was done, how it was implemented, and why specific approaches were chosen. Section 4 presents the results, including the creation of the synthetic image dataset, model training performance metrics, and inference outputs. It addresses the what of the results. Section 5 discusses these results in depth, evaluating areas for improvement and proposing potential directions for future work. It explores the why behind the findings. Finally, Section 6 concludes the report.

2 Literature review

2.1 Introduction

The main objective of this section is to present, evaluate, and discuss existing research, theories, and evidence related to space debris detection, synthetic image generation, and lightweight object detection models. This review positions the current project within a broader research context, identifying gaps that it seeks to address—particularly the scarcity of real-world space debris images, the importance of a photorealistic dataset, and the limited compute resources available for deploying ML models on the chaser.

To ensure relevance and coherence with the technical challenges explored in this project, sources were selected based on their direct contribution to one or more of the following domains: synthetic image dataset generation, photorealistic rendering for space imaging, lightweight object detection models for real-time inference, and augmentation strategies for improved model generalization. Priority was given to peer-reviewed primary research studies published between 2017 and 2024, as well as technical reports that introduced novel tools or methods relevant to on-orbit ML applications. The search process involved using Scite and Elicit, two AI-powered research assistants, to identify highly cited and topically relevant papers, followed by reviewing the references within those papers to uncover additional relevant sources.

2.2 The need for space debris detection and limitations of traditional methods

The growing concentration of debris in LEO poses a significant threat to active space missions. Current tracking systems—such as radar and ground-based optical observatories—are limited to detecting larger objects, leaving an estimated 131 million sub-centimetre fragments largely untracked [6]. Traditional vision-based detection techniques, like SIFT, have been explored for pose estimation of uncooperative and unpredictable targets [10], but these methods struggle under variable lighting and when parts of the debris are obscured. They also demand considerable computational resources, making them unsuitable for deployment on low-power edge devices.

To address these limitations, recent approaches have turned to DL models for object detection, which have shown superior performance in visually challenging scenarios [13]. Models such as Faster R-CNN and the YOLO variants have achieved high accuracy in detecting small objects [13], [14]. However, they depend on large, well-annotated datasets, and the more advanced variants require substantial GPU

resources—posing challenges for real-time operation on power and compute constrained chaser systems.

2.3 Synthetic image generation for data scarcity in space applications

Given the limited availability of real-world images of space debris, researchers have embraced synthetic image datasets as an alternative for training object detection models. Armstrong et al. [15] demonstrated that synthetic data could achieve competitive performance in semantic image segmentation of 2D images of unmanned spacecraft. However, their synthetic image dataset pipeline did not simulate camera or sensor effects, leading to consistent, and relatively low, performance during tests on unknown target spacecraft and real-world images. Similarly, Bechini et al. [16] adopted POV-ray to generate high-fidelity synthetic image datasets. However, this came at a cost of increased computational resources due to long parsing times and no GPU acceleration.

More recent work, including the SPEED+ dataset [17], emphasizes bridging the synthetic-to-real domain gap through hardware-in-the-loop validation, using robotic testbeds to produce high-fidelity images under space-like conditions. More recently, `Blender` has emerged as a viable alternative for synthetic image dataset generation [18], [19], [20], with applications ranging from asteroid boulder segmentation to resident space object (RSO) classification. These efforts demonstrate `Blender`'s flexibility in replicating realistic space conditions. Its widespread adoption can be attributed to its open-source nature, free availability, powerful Python API, advanced Cycles rendering engine, and active support community.

This project builds on this adoption by developing a synthetic image generation pipeline with features not commonly implemented in prior work, designed to be more scalable, efficient, and accessible addressing the challenges outlined in Section 1.2.

2.4 Photorealism and rendering approaches in simulation pipelines

Photorealism plays a pivotal role in narrowing the domain gap between synthetic and real images. Researchers have explored multiple rendering engines—POV-Ray [21], Mitsuba2 [21], and STK [22]—to model physical effects like shadows, surface reflections, and how light scatters through the atmosphere. While these tools offer advanced simulation capabilities, their steep learning curves, licensing constraints, or compute demands restrict widespread adoption.

Schubert et al. [18] also highlighted the importance of shadow modelling and surface reflectance for narrowing the synthetic-to-real domain gap. Their use of `Blender`'s compositor for augmentations, however, was slow and inflexible. This project adopts a more modular approach by separating rendering and augmentation stages, allowing synthetic image generation to be faster and more efficient.

Bechini et al. [16] proposed several validation metrics to assess photorealism, including pixel intensity histogram comparisons, a shadow index to quantify how accurately shadows are reproduced when compared to real-world reference images, and a feature quality index to evaluate the reliability of extracted features. Yet, such quantitative validation techniques remains rare due to the limited availability of publicly accessible real-world spaceborne images.

2.5 Deep Learning-based object detection models and lightweight alternatives

Object detection for space debris requires models that balance accuracy, speed, and hardware efficiency. Two-stage models like Faster R-CNN have been popular due to their accuracy but are unsuitable for real-time onboard inference due to their high latency [13]. In contrast, the YOLO family offers a one-stage architecture with lower inference times, making it suitable for edge deployment [14], [23].

Recent work by Xi et al [24] demonstrated improved accuracy in space object detection using a YOLO-based variant, while Guo et al. [14] introduced a cross-scale feature fusion module (CCFM) in YOLOv8 to enhance small debris detection in noise-heavy celestial scenes. These enhancements notably improved mean Average Precision (mAP), especially under low-signal-to-noise ratio conditions. Building on these developments, this project adopts YOLOv8n for its high performance and low parameter count—making it well suited for deployment on low-power edge devices like the Jetson Nano. This choice directly addresses the computational constraints highlighted in prior research and aligns with the growing shift toward onboard AI in autonomous debris removal chaser systems [17].

2.6 Domain randomization, augmentation, and generalization challenges

A key challenge with synthetic image datasets is ensuring model generalisation to real-world images. Domain randomization has proven effective in addressing this, introducing variations in camera angles, lighting, backgrounds, and object poses to prevent model overfitting. SPEED+ [17] and BoulderNet [19] applied randomized poses and lighting conditions to their datasets to improve model robustness. In contrast, Armstrong et al. [15] reported poor performance, largely due to limited variation in their dataset. One persistent issue across many pipelines is the rotational bias introduced by Euler angle sampling, which results in uneven pose distribution. This project addresses that gap by using quaternions to achieve uniform rotational coverage, mitigating orientation bias during training. Augmentation also plays a crucial role in bridging the realism gap. Y. Guo et al. [14] applied noise and blur to simulate telescope imaging artifacts. This project expands on that by incorporating eight augmentation techniques, including cosmic ray strikes and lens distortion, applied post-render via OpenCV and Albumentations.

2.7 Summary

The reviewed literature highlights both the promise and the limitations of current space debris detection strategies that rely on synthetic image datasets and object detection. This project builds on those advances while addressing key shortcomings by integrating a `Blender`-based synthetic image generation pipeline with a YOLOv8n model. The resulting contribution offers a scalable, efficient, and accurate solution for onboard space debris detection, positioning the pipeline as a high-potential and widely adaptable approach for future debris mitigation missions.

3 Methods

3.1 Method Selection and Justification

3.1.1 Synthetic Image Generation

The first key challenge in applying DL to space debris detection is the need for large, photorealistic, and diverse datasets. As discussed in Section 1, acquiring such datasets is difficult because of the uncooperative and unpredictable nature of space debris, as well as the high costs and long timescales associated with preliminary inspection missions. Consequently, there is a severe lack of real images of existing space debris and spacecraft, and nothing at all for spacecraft that are yet to launch.

To address this, synthetic image generation has emerged as a viable solution. Synthetic images are artificially generated that are almost exact facsimiles of real-world iamges, enabling training object models without relying on scarce real data. In this project, a 3D computer graphics software (Blender, Ver. 4.3.2; Blender Foundation, Amsterdam, Netherlands) was used to generate a synthetic image dataset.

`Blender` was chosen for four primary reasons:

1. **Ease of use and accessibility:** `Blender` is widely used, has an extensive community, and is 100% free for both commercial and personal use. This ensures that the synthetic image generation pipeline is easily replicable by other researchers. Other 3D modeling software like Unreal Engine, Autodesk Maya, and NVIDIA Omniverse have steeper learning curves, licensing restrictions, or hardware dependencies, making them less universally accessible.
2. **Capability to achieve key characteristics of a synthetic image dataset:** `Blender` provides the tools necessary to generate a photorealistic, diverse, and well-labelled synthetic image dataset.
3. **Advanced scripting capabilities:** `Blender` has one of the most robust Python APIs among 3D modeling and rendering software, enabling full automation of dataset generation. While Unreal Engine, another widely used 3D modeling tool, is primarily designed for game development, it offers scripting capabilities through its visual scripting language, Blueprint. However, Blueprint lacks the flexibility of `Blender`'s Python API and can become cumbersome and complex, particularly for large-scale and advanced projects like this. Moreover, with recent advancements in the ability of large-languge models to generate highly accurate code, it further strengthens the need to use `Blender` to fully leverage this capability.
4. **GPU-accelerated rendering:** `Blender` natively supports Compute Unified Device Architecture (CUDA) and NVIDIA OptiX, enabling faster, more efficient rendering on NVIDIA GPUs. Without GPU acceleration, rendering in `Blender` can be up to seven times slower, depending on hardware and scene complexity. For instance, rendering a single image for this project's synthetic image dataset took approximately 37 seconds using GPU + CUDA, compared to 240 seconds with a CPU.

The following key characteristics were essential for the synthetic image dataset and were directly supported by `Blender`.

1. **Diversity:** Diversity is crucial for training robust ML models, as it exposes them to different debris types, orbital conditions, and common space-based imaging artifacts. `Blender` satisfies this requirement through its Python API, which enables automated dataset generation by randomizing parameters such as debris orientation, camera positioning, debris location, cloud cover, and sun illumination [18].
2. **Photorealism:** A photorealistic synthetic image dataset ensures that space debris is realistically represented by capturing variations in shape, material, lighting conditions, and chaser sensor characteristics. The ability of object detection models to generalize to unseen images depends on minimizing the "reality gap" between synthetic and real images [18]. Achieving photorealism is essential for ensuring that models trained on synthetic images can perform effectively in real-world scenarios. `Blender` satisfies this requirement through its Cycles rendering engine, a physically based ray-tracing renderer [25]. Cycles ensures photorealism by accurately simulating light interactions in the space environment, applying physically based shading to replicate the reflective and absorptive properties of debris materials, and utilizing High Dynamic Range Image (HDRI) lighting to mimic the harsh illumination of the Sun and the deep shadows cast in space.
3. **Accurate labelling:** The scarcity of real images also leads to a lack of labelled training data, which is essential for supervised learning algorithms. Labels may include bounding boxes, segmentation masks, depth maps, and surface normals. `Blender` satisfies this requirement through its Python API, Compositor, and Render Passes, allowing automated generation of multiple annotation types during rendering.
4. **Simulated space environment:** With increasing interest in long-term lunar missions and future Mars exploration, considering space debris beyond Earth's orbit is relevant. `Blender` satisfies this requirement by enabling dynamic backgrounds for Earth, the Moon, and Mars. Additionally, by utilizing drivers and `constraints`, `Blender` can approximate Keplerian orbits, simulate orbital motion, and model interactions with planetary surfaces, making it a suitable tool for generating realistic space debris scenarios.
5. **Flexibility and adaptability:** A synthetic image generation pipeline must be adaptable to different debris types (e.g., defunct satellites, spent rocket bodies, small fragments) and various ADR mission scenarios. `Blender`'s Python API allows full control over image generation parameters, making it easy to produce images of different debris types under varying conditions. Given a suitable 3D debris model, `Blender` can generate images with any combination of predefined parameters, ensuring flexibility across multiple use cases.
6. **Low cost and scalability:** Capturing real images of space debris through a space mission is prohibitively expensive. For instance, the ELSA-d mission, a recent ADR demonstration, likely incurred costs between $50 million and $100 million [26], making real space-based imagery among the most expensive to acquire. `Blender` satisfies this requirement by being entirely free, with costs limited to compute resources such as cloud-based render farms or local GPUs. Additionally, `Blender` supports batch rendering and can run headless (without a graphical user interface), enabling efficient rendering on cloud servers or local GPUs. This allows for scalable, cost-effective generation of large synthetic image datasets.

By using `Blender`'s suite of advanced rendering capabilities and automation tools, this project developed a synthetic image generation pipeline that produces high-quality, annotated space debris images. This approach enables the use of synthetic data to train object detection models without having to rely on expensive real-world image acquisition.

3.1.2 Machine Learning Model Development

The extensive references to synthetic image datasets in Section 3.2.1 pertain to the training of DL-based object detection models, a subset of ML. In the context of space debris, researchers worldwide use synthetic image datasets to train object detection models for real-time, high-accuracy tracking of debris in Earth's orbit—particularly LEO. This is essential because a large part of space debris consists of tiny fragments that are very difficult to detect [27]. The existing ensemble of radar and radio telescopes primarily track larger objects, which, while still hazardous, are less elusive and arguably pose a lower risk of high-speed collisions than smaller fragments [27].

The second key challenge in applying DL to space debris detection is the operational constraints of the chaser, namely: power, weight, and computational capacity. As discussed in Section 1, many legacy models rely on computationally voracious GPUs for training and inference, making deployment on a chaser difficult due to the mentioned constraints. To address this, the widely adopted You Only Look Once (YOLO) family of models has emerged as a viable solution, offering fast, real-time, and accurate space debris detection while remaining computationally lightweight.

Unlike traditional two-stage object detection models such as Faster Region-Based Convolutional Neural Networks (Faster R-CNN), YOLO processes an entire image in a single forward pass —looks at it 'Only Once'—significantly improving detection speed. By using a single neural network to identify objects and determine their locations within an image, YOLO achieves real-time detection with high accuracy. This has made it a standard choice for various applications, including autonomous driving, security surveillance, and of course, space debris tracking. Off-the-shelf object detection models like YOLO outperform conventional target detection approaches, as demonstrated in [27]. In this project, YOLOv8, an iteration in the YOLO family of models, was used.

YOLOv8 was chosen for two primary reasons:

1. **Accuracy and speed:** Object detection models are commonly evaluated based on mean Average Precision (mAP) and inference latency. The mAP provides a single-score metric reflecting a model's ability to detect objects accurately, without explicitly considering localization precision. Inference latency, measured in milliseconds, represents the total time required for preprocessing, inference, and post-processing. Table 1 highlights YOLOv8's superior performance compared to other widely used object detection models [28]. With the highest mAP@50 score of 0.62, YOLOv8 demonstrates enhanced detection accuracy, while its low inference latency of 1.3 ms enables faster image processing than its counterparts. This low latency is critical for the chaser, where real-time debris detection and tracking are essential for precise trajectory adjustments.

Table 1. Performance Metrics of Popular Object Detection Models, Including mAP@50 and Inference Latency.

Model	mAP@50	Inference Latency (ms)
YOLOv8	0.62	1.3
Faster R-CNN	0.41	54
EfficientNet	0.47	N/A
YOLOv5	0.58	N/A

2. **Computationally lightweight:** YOLOv8 is available in multiple sizes—nano, small, medium, large, and extra-large—each varying in parameter count. While more parameters improve learning capacity, they also increase computational demands. It is worth noting that Faster R-CNN variants have parameter counts similar to larger YOLOv8 models. Table 2 presents the parameter counts for all YOLOv8 sizes [29]. For space debris detection, YOLOv8n or YOLOv8s would be the best choices due to their lightweight nature, making them ideal for low-powered edge devices. These devices will be crucial components of future chasers, which must operate within strict power, weight, and computational budgets.

Table 2. Parameter Count (in Millions) for YOLOv8 Model Variants.

Model	Parameter Count (Millions)
YOLOv8n	3.2
YOLOv8s	11.2
YOLOv8m	25.9
YOLOv8l	43.7
YOLOv8x	68.2

These factors make YOLOv8 a strong choice for on-orbit servicing, docking, and ADR missions, where real-time vision-based navigation is essential.

3.1.3 Programming and Implementation

With `Blender` selected for synthetic image dataset generation and YOLOv8n as the object detection model, an Integrated Development Environment (IDE) was required to integrate these components into a cohesive space debris detection pipeline. For this project, Visual Studio Code (VS Code, Ver. 1.98; Microsoft Corporation, Redmond, WA, USA) was used for development and implementation.

VS Code was chosen for three primary reasons:

1. **Industry-leading IDE with widespread adoption:** VS Code is the most widely used IDE among developers. According to a Stack Overflow survey (2024), 73.6% of developers used VS Code—more than twice as many as its closest alternative, Visual Studio [30].
2. **Seamless integration with Anaconda for package management:** Anaconda, a Python distribution tailored for ML and data science, provides an integrated package manager (Conda), an environment manager, and over 1,500 pre-installed ML and data science packages. It enables the

creation of isolated environments, preventing dependency conflicts between projects [31]. VS Code integrates seamlessly with Anaconda, simplifying package and environment management and ensured a structured Python development environment for this project.

3. **AI-powered coding assistance with GitHub Copilot:** VS Code supports GitHub Copilot, an AI-driven coding assistant that accelerated development by providing real-time code suggestions, debugging hints, and automated boilerplate code generation. This feature streamlined the development of the space debris detection pipeline explored in this project.

These factors, along with the Jupyter Notebook extension in VS Code [32], allowed the development of a modular, understandable, and reproducible space debris detection pipeline. The main structure of this pipeline consists of image preprocessing (scaling and data augmentation)→model training→model performance analysis→inference.

1. **Preprocessing – scaling:** The images produced by the synthetic image generation pipeline are scaled by a script from 1080 ×1080 to 640 ×640 to match YOLOv8n's default training and inference size.

   This preprocessing step is not necessary, as YOLOv8n automatically resizes input images to the specified `imgsz` before training and inference. However, it is performed to maintain greater control over preprocessing, ensuring consistent quality in the synthetic image dataset. While YOLOv8n does not strictly require 640 ×640, this resolution is the default as it provides an optimal balance of detection accuracy, inference speed, and computational load.

   Larger images (e.g., 1080 ×1080) improve the detection accuracy of small debris fragments but increase computational load, slowing down training and inference. This higher computational demand makes them unsuitable for low-power edge devices.

   Conversely, smaller images (e.g., 416 ×416) reduce computational load and speed up training and inference but compromise detection accuracy for small debris fragments. This reduced detection accuracy makes them less effective for ADR missions.

2. **Preprocessing – data augmentation:** The images produced by the synthetic image generation pipeline are augmented by a script that applies one of Gaussian noise, random occlusion, motion blur, grayscale conversion, vignetting, lens distortion, Poisson noise, or simulated cosmic ray strikes. Data augmentation artificially increases dataset variability by modifying existing images [33]. It improves a ML model's ability to generalize by introducing transformations that mimic real-world conditions [34]. While augmentation is beneficial for preventing overfitting and compensating for limited training data, any biases in the original dataset are inherently transferred to the augmented images.

   Each augmentation technique was selected to simulate specific real-world conditions observed in space debris imagery:

   - **Gaussian noise:** Real-world images of space debris often contain sensor noise due to low-light conditions, cosmic radiation, and electronic interference in cameras.
   - **Random occlusion:** Space debris images frequently have missing or obscured regions caused by spacecraft structures, partial field-of-view limitations, debris self-occlusion during tumbling, or

missing pixel data due to sensor failures on the chaser.

- **Motion blur:** Motion blur occurs due to relative motion between the chaser and the debris, which may result from tumbling debris, tracking errors, or vibrations from thruster adjustments.
- **Grayscale conversion:** Many space debris images are captured using monochrome sensors found in navigation cameras, LiDAR systems, and infrared imaging, making grayscale conversion a relevant augmentation.
- **Vignetting:** Spaceborne cameras often experience uneven illumination and lens shading effects due to optical limitations and varying light conditions in space.
- **Lens distortion:** Wide angle lenses, commonly used in proximity operations and VBN, introduce perspective distortions and warping in real-world images of space debris.
- **Poisson noise:** Space debris images often exhibit photon shot noise, which results from random fluctuations in the number of photons reaching a sensor. This effect is more pronounced in low-light environments and is independent of imaging electronics.
- **Simulated cosmic ray strikes:** High-energy cosmic rays impact camera sensors, producing bright spots or streaks in images, a common occurrence in space-borne imaging.

By applying these augmentations, the synthetic image dataset better reflects the challenges of real-world space debris detection, improving YOLOv8n's robustness to variations encountered during inference (see Fig.10).

3.1.4 Model Evaluation Techniques

ML is transforming the space industry, but sustained progress depends on rigorous model evaluation [35]. In this project, evaluating the YOLOv8n model is critical to determining whether it generalizes well when trained on synthetic images and tested on unseen synthetic or real-world debris images. Model performance can be assessed using several techniques, including cross-validation, test set evaluation from a train-validation-test split, and testing on real-world debris images. If applied to this project, these techniques would be implemented as follows:

1. **Cross-validation:** To prevent overfitting and improve generalizability, cross-validation would ensure that YOLOv8n does not rely excessively on specific features within the synthetic image dataset and performs consistently across different subsets. The dataset would be divided into k subsets, with the model trained on k-1 subsets while the remaining subset serves as the test set. This process would be repeated k times, with each subset acting as the test set once. By averaging performance across all iterations, cross-validation would provide a more reliable measure of the model's adaptability within the synthetic image dataset.
2. **Train-validation-test split:** To establish a structured training process and ensure objective performance evaluation, the synthetic image dataset would be divided into three sets. The training set would be used to teach YOLOv8n to detect space debris by learning patterns in synthetic images.

The validation set would be used to fine-tune hyperparameters and monitor performance to prevent overfitting. The test set would then provide an unbiased assessment of the model's detection capability on unseen synthetic images. This split creates a controlled environment for evaluating model performance while keeping the test images independent of the training process.

3. **Evaluation on real-world debris images:** After training on the synthetic image dataset, testing the model on real-world debris images would assess its ability to generalize beyond synthetic data. This evaluation would determine how well the patterns learned from synthetic images transfer to real-world conditions, where factors such as lighting variations, sensor noise, and irregular debris shapes introduce additional complexity. The results would indicate the extent to which the synthetic image dataset effectively contributed to training a robust space debris detection model.

In this project, the train-validation-test split was chosen as the primary evaluation technique for YOLOv8n due to its efficiency and straightforward implementation. Evaluation on real-world space debris images was also carried out; however, this was limited due to the scarcity of such images—an issue that serves as one of the primary motivations for this project. Cross-validation was not used, as it is significantly more computationally expensive and time-consuming compared to the other two techniques. Furthermore, given the synthetic image generation pipeline's capability to produce sufficiently large datasets, cross-validation was deemed unnecessary.

3.1.5 Model Performance Metrics

Model evaluation typically relies on quantitative measures, though qualitative assessments may occasionally be useful. This project focuses exclusively on quantitative metrics, as they provide clear, reproducible, and comparable results. The key performance metrics used to assess the YOLOv8n space debris detection model were Intersection over Union (IoU), Precision, Recall, Average Precision (AP), and mAP.

1. **Intersection over Union:** IoU measures the overlap between a predicted bounding box and the ground truth bounding box, as mathematically defined in (1). The IoU threshold specifies the minimum required overlap for a prediction to be considered correct. Based on this threshold, predictions are classified into True Positives (TP), False Positives (FP), and False Negatives (FN), which form the basis of the confusion matrix. If IoU $\geq$ IoU threshold, the prediction is counted as a TP; otherwise, it is considered an FP.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}} \tag{1}$$

- **YOLOv8n Processing and Thresholding:** The YOLO family of models processes an entire image in a single forward pass through the neural network. This efficiency is achieved through its grid-based architecture, where an image is divided into grid cells, and each cell predicts multiple bounding boxes along with class probabilities. Each predicted bounding box is assigned a confidence score, representing the model's certainty that the box contains an object. To refine these predictions, two filtering mechanisms are applied: confidence thresholding and

Non-Maximum Suppression (NMS).

  - **Confidence Thresholding:** Bounding boxes with confidence scores below the confidence threshold are removed, retaining only the strongest predictions.
  - **Non-Maximum Suppression (NMS):** Even after confidence filtering, multiple bounding boxes may predict the same object. The IoU is calculated for the remaining boxes. If two boxes have an IoU greater than the IoU threshold, NMS removes the one with the lower confidence score. This ensures that only the most confident bounding box remains for each detected object.

- **Confusion Matrix:** A confusion matrix is a table that visually summarizes the performance of a classification model by comparing its predictions with the ground truth [36]. The term confusion matrix has two sources of origin: first, it identifies instances where a model confuses one class for another; second, its structure can sometimes cause confusion for readers.

- **Class of Predictions:** Predictions are classified based on their IoU value with the ground truth bounding box:
  - **TP:** The model predicts a bounding box around a piece of space debris, and it sufficiently overlaps with the ground truth bounding box (IoU $\geq$ IoU threshold).
  - **FP:** The model predicts a bounding box where no space debris exists, or the predicted bounding box has insufficient overlap with the ground truth bounding box (IoU $<$ IoU threshold).
  - **FN:** The model fails to predict a bounding box around a piece of space debris that is present in the image.
  - **TN:** The model does not predict a bounding box because no space debris is present in the image. TN is not typically used in object detection but conceptually refers to correctly identifying the absence of an object where none exists.

2. **Precision and Recall:** Precision and recall measure different aspects of the model's performance:

   - Precision quantifies the accuracy of the model's positive predictions by measuring the proportion of correctly predicted positive objects (TP) out of all predicted positives (TP + FP). It is mathematically defined in (2). Precision depends on the IoU threshold, as this threshold determines whether a predicted object is classified as TP or FP.

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

   - Recall assesses the model's ability to detect all actual positive objects by measuring the proportion of correctly predicted positives (TP) out of all actual positives (TP + FN). It is mathematically defined in (3). Like precision, recall is influenced by the IoU threshold, as it affects whether a predicted object is classified as TP or FN.

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

3. **Precision-Recall Curve:** Given the importance of both precision and recall, the Precision-Recall (PR) curve provides a comprehensive way to analyze their relationship. The PR curve is obtained by plotting a model's precision against recall at varying confidence score thresholds while keeping the IoU threshold fixed [37]. Consequently, different IoU thresholds yield different PR curves.

   A key question arises: Why is a PR curve necessary instead of evaluating precision and recall separately? A model with high precision confidently identifies positive detections, meaning it minimizes FPs. Meanwhile, a model with high recall detects most positive objects, ensuring that few TPs are missed. However, these metrics often trade off against each other. A model with high recall but low precision detects most positive objects but also generates numerous FPs, misclassifying negative objects as positive. Conversely, a model with high precision but low recall makes highly accurate positive classifications but fails to detect many actual positive objects. The PR curve visualizes this trade-off, enabling a more informed selection of the best confidence score threshold to balance precision and recall based on the specific requirements of the application.

4. **Average Precision:** AP is a widely used performance metric in object detection that evaluates model performance on a per-class basis. Defined as the area under the PR curve, it quantifies how well a model balances precision and recall across varying confidence score thresholds. AP condenses the PR curve into a single scalar value, providing an overall measure of detection quality. A high AP indicates strong performance, where both precision and recall remain consistently high across different confidence thresholds, while a low AP suggests a significant drop in either precision or recall. AP values range from 0 to 1, with 1 representing perfect detection performance. The general formulation of AP is provided in (4), while (5) presents a common method for calculating the area under the PR curve to determine AP.

$$\text{Average Precision} = \int_{r=0}^{1} p(r)\, dr \tag{4}$$

$$\text{Average Precision} = \sum_{k=1}^{n-1} [r(k) - r(k-1)] \times \max\left(p(k), p(k-1)\right) \tag{5}$$

5. **Mean Average Precision:** mAP extends AP by evaluating model performance across multiple classes. It is computed as the average AP over all classes under consideration [38]. The mathematical formulation of mAP is provided in (6).

$$\text{Mean Average Precision} = \frac{1}{k} \sum_{i}^{k} AP_i \tag{6}$$

   mAP is the most comprehensive and widely adopted performance metric in object detection, encapsulating both detection accuracy and localization quality across multiple classes. There are two variants of mAP: mAP@50 and mAP@50:95.

   - **mAP@50:** mAP calculated using a fixed IoU threshold of 0.50. A detection is classified as a TP if the IoU between the predicted and ground truth bounding boxes is at least 50%. Since this threshold allows for some imprecision in localization, mAP@50 is considered a more lenient

evaluation metric, enabling models to achieve high scores even if their predicted bounding boxes do not perfectly align with the ground truth.

- **mAP@[50:95]:** mAP computed by averaging over multiple IoU thresholds, ranging from 0.50 to 0.95 in increments of 0.05. Unlike mAP@50, this metric evaluates model performance across a spectrum of localization accuracies. As higher IoU thresholds require greater alignment between predicted and ground truth bounding boxes, mAP@[50:95] is a stricter evaluation metric.

This is why fundamental concepts such as IoU, TP, FP, FN, TN, recall, precision, PR curve, and AP have been discussed—together, they form the foundation upon which mAP is built, making it the standard benchmark for evaluating object detection models. Fig. 1 shows a flowchart illustrating the computation of mAP for object detection.

3.1.6 Logging and Monitoring Model Performance:

3.2 Synthetic Image Dataset Scene Setup

3.2.1 Earth and Sun

Modelling the Earth in `Blender` presents unique challenges due to its complex surface features, which include vast landmasses with varying colors and extensive bodies of water that exhibit both specular reflection and refraction. To ensure the synthetic image dataset maintained photorealism, it was essential to create a realistic Earth model. This was achieved using high-resolution Earth textures provided by NASA [39]-[40], as shown in Fig. 2.

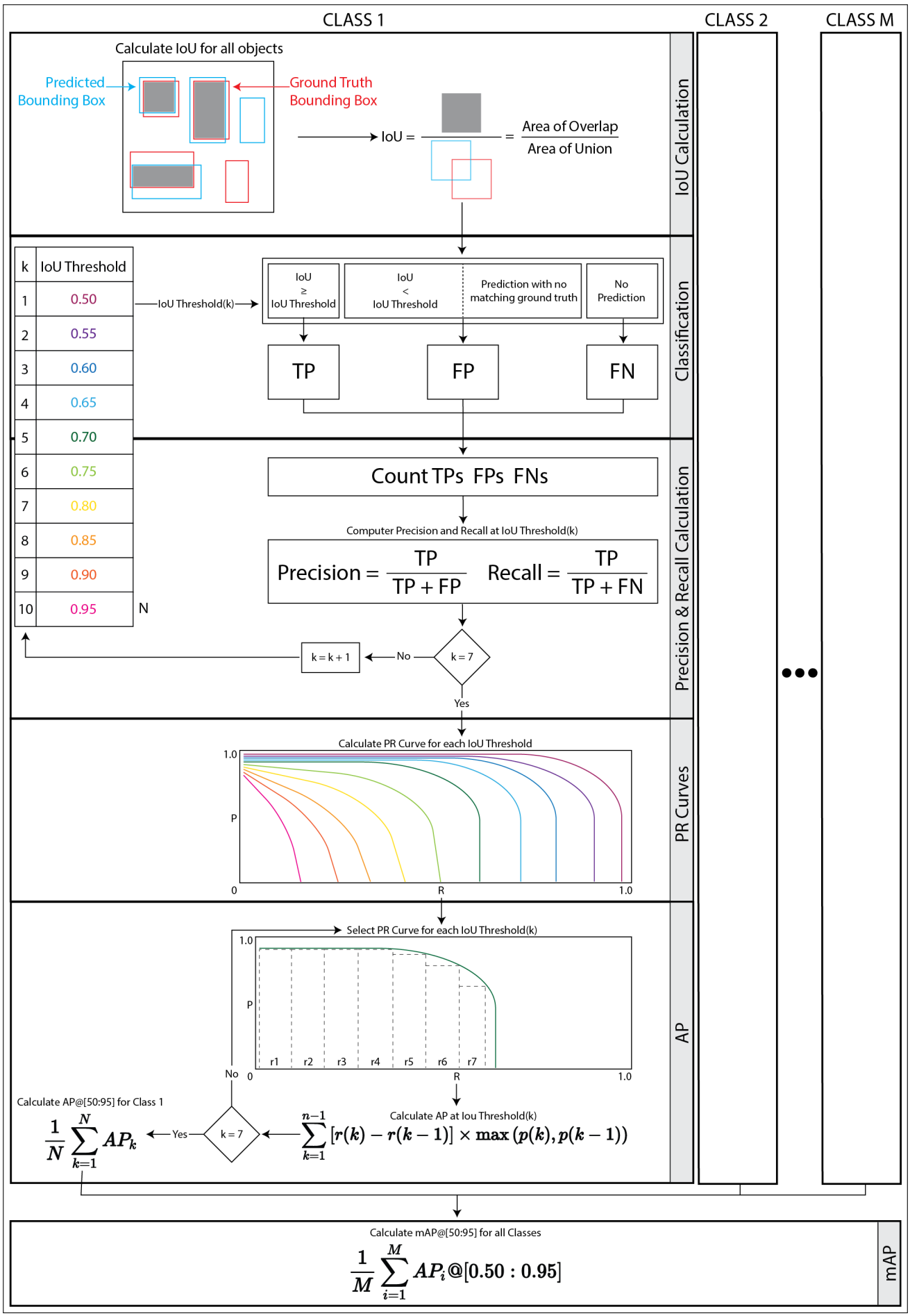
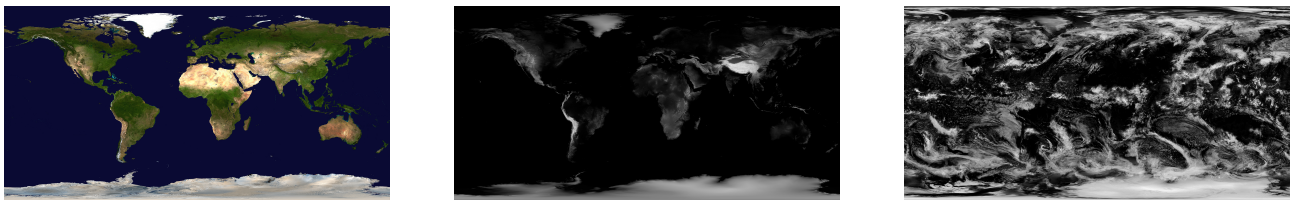
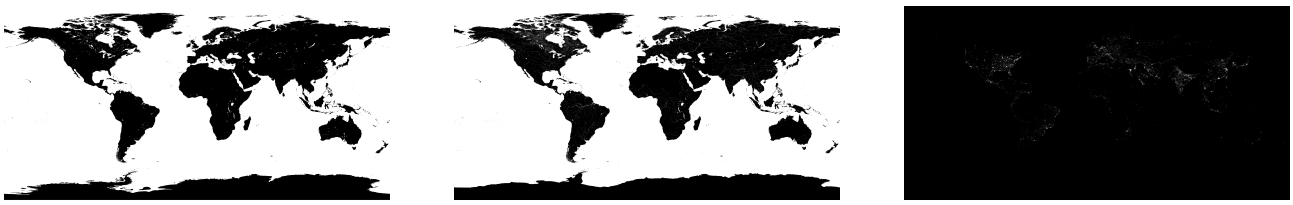


Fig. 1. Flowchart illustrating the computation of mAP@[0.50:0.95].



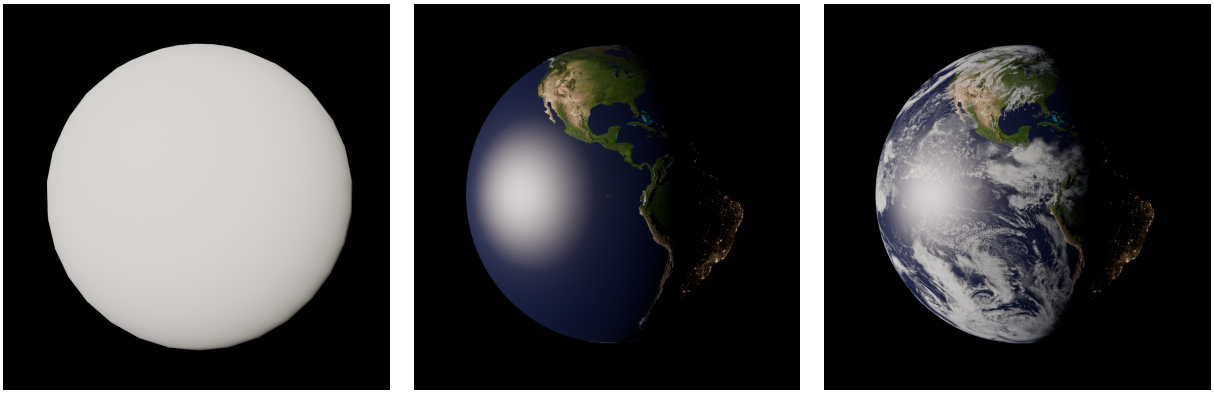
(a) Albedo Map **(b)** Topography Map **(c)** Cloud Map



(d) Specular Map **(e)** Land and Ocean Mask **(f)** Night Lights Map

Fig. 2. NASA Earth texture maps for realistic rendering

The modelling process used these textures alongside established displacement techniques, adaptive subdivision, material configurations, and compositing methods to achieve photorealistic results in `Blender` [41]. In reality, the Earth's radius is 6,378,000 m, but the `UV Sphere` used in `Blender` to model the Earth had a radius of 159.45 m—making the scale 40,000:1. This scale was chosen arbitrarily, with the primary consideration being rendering time, as the time required to generate each image would have increased in proportion to the dimensions of the `UV Sphere`. An overview of this process, from an initial `UV Sphere` to the final Earth model, is presented in Fig. 3.



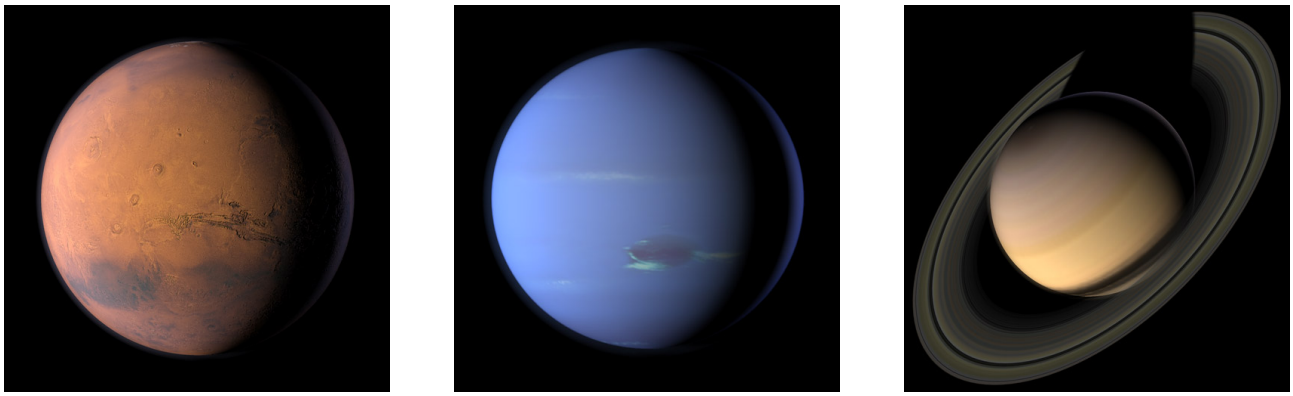
(a) UV Sphere **(b)** Day–night terminator and ocean specular reflection. **(c)** Volumetric clouds complete the realistic planetary appearance.

Fig. 3. Progressive Earth rendering in Blender, illustrating the addition of surface color, atmosphere, and clouds.

The scene's primary light source, the Sun, was incorporated by adding a `Sun` light (`Add`→`Light`→`Sun`) and positioning it at the Earth's center. The properties of the `Sun` light were adjusted in the `Object Data Properties` panel. The `Strength` was set to `8 W/m²`, reflecting the Sun's irradiance as measured in `Blender`'s physically based lighting system. The `Angle` was set to `0.526°`, matching the Sun's apparent angular diameter as seen from Earth. The color was set to white, as sunlight contains all visible wavelengths, and in space, without atmospheric scattering, it appears white, ensuring an accurate representation.

While space debris is currently an issue confined to Earth's orbit, the increasing human presence in space over the coming decades and centuries could result in similar challenges around other planets. Discarded

landers, decommissioned satellites, and spent propulsion stages could accumulate in stable orbits around celestial bodies, particularly Mars, which is expected to see an increasing number of robotic and crewed missions. Gas giants such as Saturn and Neptune, while lacking solid surfaces, have extensive ring systems and moons where debris from exploratory missions could persist. Creating realistic planetary models is therefore not only useful for rendering the Earth but also for simulating conditions on other planets to study the behavior of objects in orbit, atmospheric interactions, and planetary illumination. By replacing the high-resolution Earth textures with those of other planets [39], the model can be adapted to generate photorealistic renders of Mars, Saturn, Neptune, and beyond. Three examples of such adaptations are shown in Fig. 4 [39].



(a) Mars. **(b)** Neptune. **(c)** Saturn.

Fig. 4. High-fidelity renders of Mars, Neptune, and Saturn

UV Pinching at the poles:
Problem: While modeling the Earth in `Blender`, a graphical artifact occurred due to the default UV mapping of the `UV Sphere` used for the planet's geometry. The UV map compressed large portions of a 2D texture into small triangular faces at the poles, causing noticeable pinching and distortion. This effect was particularly pronounced with high-resolution Earth textures. **Solution:** To eliminate the pinching effect, the UV mapping was modified in `Blender` using the shader editor. With the `Node Wrangler` add-on enabled, a `Texture Coordinate` node and a `Mapping` node were added to each `Image Texture` by selecting it and pressing `CTRL + T`. The coordinate output was switched from `UV` to `Generated`, and the `Image Texture` projection method was set to `Sphere` instead of the default UV mapping. These adjustments were applied to all four `Image Texture` nodes in the `Surface` shader and the `Image Texture` node in the `Clouds` shader, successfully removing visible seams at the poles and improving the `Blender` model's accuracy. For further reference, a discussion on this issue is available in [42].

3.2.2 LEO Geometry

LEO has long been established as an orbital space junkyard [43], with 40,500 objects larger than 10 cm, 1.1 million objects between 1 cm and 10 cm, and approximately 131 million fragments ranging from 1 mm to 1 cm, and counting [44]. Consequently, all space debris modeled in the synthetic image dataset for this project were assumed to be in orbit at 560 km altitude. A detailed procedure for modeling the LEO geometry is described in Tables A.1–A.7. Fig. 5 illustrates how the LEO geometry would appear

when modeled in Blender.

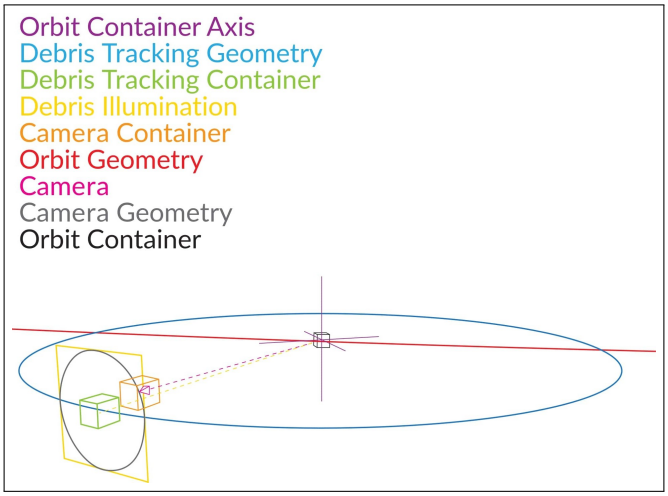


Fig. 5. LEO geometry representation in Blender with key components labeled.

3.2.3 Space Debris

The 4 classes of space debris picked for the synthetic image dataset were:

- **CubeSat**: CubeSats are increasingly being deployed for research and commercial applications, contributing to the growing population of objects in LEO. Their small size makes them difficult to track, elevating the risk of collisions with other satellites and debris. Additionally, the lack of onboard propulsion in many CubeSats prevents controlled de-orbiting, resulting in prolonged orbital lifetimes and exacerbating long-term space congestion. This debris class is labeled as `Satellite` in the .blend file, as referenced in [45].

- **Envisat**: Weighing approximately 7800 kg, Envisat is one of the largest uncontrolled objects in LEO [46]. Due to its considerable size and orbital altitude of 766 km, it presents a high-risk debris source, as a collision would generate a large debris cloud. Given its altitude, Envisat is expected to remain in orbit for centuries, making it a persistent long-term hazard. Additionally, its orbital path intersects with key satellite constellations, further increasing the probability of a collision [46]. This debris class is labeled as `Envisat` in the .blend file, as referenced in [45].

- **Hubble**: Weighing over 11,000 kg, Hubble is one of the heaviest objects in LEO. With no further servicing missions planned, it is expected to eventually become a derelict object if no intervention occurs, posing a long-term risk to the orbital environment. However, NASA has explored potential options for controlled de-orbiting or re-boosting to mitigate this risk [47]. This debris class is labeled as `Hubble` in the .blend file, as referenced in [45].

- **Falcon 9 2nd Stage**: At the time of writing, SpaceX was launching approximately 13–15 Falcon 9 rockets per month, making it the most frequently used launch vehicle [48]. While most Falcon 9 second stages and payload fairings are actively de-orbited by SpaceX [49], this debris class was selected due to the prevalence of rocket bodies, particularly upper stages, among the statistically most concerning derelict objects in LEO [46]. This debris class is labeled as `Falcon 9 F&S` in the .blend file, as referenced in [45].

A detailed procedure for adding space debris to the `Blender` scene to complete the synthetic image dataset model is described in Table A.12. The method outlined in Table A.12 is for a single debris class, `Envisat`, but was replicated to incorporate multiple classes into the `Blender` scene.

Fig. 6 illustrates how the LEO geometry will appear when modeled in `Blender` with a single debris class, `Envisat`, added.

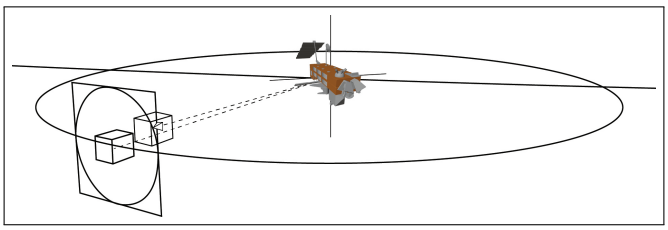


Fig. 6. LEO geometry representation in Blender with a single debris class, `Envisat`, added.

3.3 End-to-End Space Debris Detection Pipeline

The space debris detection pipeline developed for this project is an end-to-end script that can be executed either sequentially within a Jupyter notebook or as a standalone Python file [45]. Running the pipeline in a notebook supports modularity, making it easier to implement and test changes, whereas the standalone version enables full automation once the synthetic image dataset `Blender` script [45], directory structure, and parameters have been configured. The script has been designed with a strong emphasis on clarity, reproducibility, and, most importantly, scalability. At a high level, its structure is as follows:

1. **Configure directories and parameters**: Centralizes path definitions and parameter settings to maintain consistency across all stages of the pipeline.
2. **Generate synthetic images using `Blender`**: Automates labelled image generation by rendering synthetic scenes with `Blender`.
3. **Resize and convert images**: Converts and resizes synthetic images to standardize their format for YOLOv8n compatibility.
4. **Visualize bounding boxes**: Overlays bounding boxes on images to validate annotation accuracy before training.
5. **Apply data augmentations**: Applies random image distortions to enhance model robustness and generalization.
6. **Train YOLOv8n model**: Trains a YOLOv8n model while tracking progress and performance using Weights & Biases.
7. **Run inference**: Runs inference on unseen synthetic and real-world images to evaluate the model's overall performance.
8. **Load, process, and visualize spatial metadata from the `Blender` scene**: Extracts coordinate and orientation data and visualizes spatial patterns.

3.3.1 Pseudocode

The pseudocode in Tables B.13–B.20 capture the core logic of the pipeline in a clear, environment-agnostic manner. It emphasizes the problem itself rather than language-specific idiosyncrasies, making it easy to interpret and implement in code.

Full Pipeline Overview:

1. (Optional) Restart `Blender` in a loop if needed.
2. Resize and scale the synthetic images (and update labels).
3. (Optional) Visualize bounding boxes on original images.
4. Randomly apply augmentations to scaled synthetic images.
5. Train the YOLOv8 model and log results to W&B.
6. Run inference with trained weights on test images.
7. Load and plot coordinate data for further analysis (3D scatter, offset plots, etc.).

3.3.2 Key Functions and Settings

The synthetic image generation script and the space debris detection pipeline incorporate one key function and specific settings in the .blend file [45] that ensure reproducibility and scalability across a wide range of computing environments, provided a discrete GPU is available. All synthetic images for this project were rendered using an NVIDIA GeForce GTX 1070 (8 GB GDDR5), demonstrating the pipeline's compatibility with consumer-grade hardware. The function and it's purpose are outlined below:

- **`restart_blender(delay=10)`**: This function, shown in Table B.13, appears in Cell 3 of the `space_debris_detection_pipeline.ipynb` notebook. To automate the rendering and annotation of synthetic images, the synthetic image generation script (Code 1) and the `restart_blender(delay=10)` function (Code 2) were designed to work in tandem. Code 1 runs inside `Blender` and is responsible for rendering and annotating images in discrete batches. It loads a JavaScript Object Notation (JSON) file to track the current batch, renders X images per class per batch, updates the state file, and, after completing Y batches, writes a "stop flag" file to disk. Code 2, which runs outside `Blender`, orchestrates the overall pipeline by launching `Blender` in headless mode to execute Code 1. After each run, it checks for the presence of the stop flag. If the flag is not found, it restarts `Blender` to initiate the next batch. Once the stop flag is detected, indicating that all batches have been processed, Code 2 halts further execution.

  Restarting `Blender` between batches ensures that GPU memory is fully cleared between runs. This was necessary because `Blender` does not always release memory after rendering, which was observed during development—`Blender` consistently crashed around the 1360-image mark due to gradual memory buildup. By implementing this restart mechanism, the system avoids memory-related crashes, enabling stable and scalable on-demand generation of synthetic images. E.g., (100 images x 4 classes) x 6 batches = 2400 images.

Specific settings were modified in the .blend file [45], resulting in a significant reduction in rendering time per image—from an average of 1 minute 40 seconds to 14 seconds. The implemented settings

were:

- Enabled `Render Properties`→`Denoise`.
- Reduced `Render Properties`→`Max Samples` from `1024` to `256`.
- Lowered `Output`→`Resolution` from `2048`×`2048` to `1080`×`1080`.
- Increased `Render Properties`→`Noise Threshold` from `0.200` to `0.500`.
- Set `Render Properties`→`Render Engine` to `Cycles`, with →`Devices` set to `GPU`.
- For non-RTX GPUs, set `Edit`→`Preferences`→`System`→`Cycles Render Devices` to `CUDA`.
- For RTX GPUs, set `Edit`→`Preferences`→`System`→`Cycles Render Devices` to `OptiX`.

Specific settings were modified in the .blend file [45] to prevent `Blender` from crashing during the image rendering process.

- Reduced `Render Properties`→`Tile Size` to `512`. This increases render time slightly but significantly reduces GPU memory usage per render.
- Unchecked `Render Properties`→`Final Render`→`Persistent Data`. Disabling this prevents the accumulation of render data in memory—normally used for faster re-renders—which helps avoid crashes on systems with limited RAM and VRAM.

4 Results

4.1 Synthetic Image Dataset Creation

This section presents results related to two primary characteristics of the synthetic image dataset: domain randomization and photorealism. Domain randomization is evaluated using quantitative non-textual elements, while photorealism is illustrated through sample images from the synthetic image dataset. Quantitative results were not obtained for photorealistic evaluation due to the lack of corresponding real-world images for comparison. A total of 1,200 images were generated, containing an equal number of instances from each of the four debris classes. Randomization was applied to debris orientation, camera positions relative to the debris, debris positions relative to the Earth, cloud cover, and sun illumination. The dataset was then augmented with an additional 1,200 images using randomly applied image distortions through the function shown in Table B.16.

4.2 Model Training, Evaluation, and Inference

This section presents two industry-standard object detection metrics—mAP@50 and mAP@[50:95]—evaluated for three state-of-the-art versions of the YOLO model family: YOLOv8n, YOLOv11n, and YOLOv12n. YOLOv8n was used as the primary model in this project. A validation dataset consisting of 240 images (10% of the training dataset size) was used to evaluate model

performance at the end of each training epoch. During training, each YOLO model ran inference on this validation dataset to assess its performance progression. The mAP results for YOLOv8n were compared with those from the other two models to provide reference context. Fig. 11 shows a line graph with epochs on the x-axis and mAP@50 values on the y-axis for the three models.

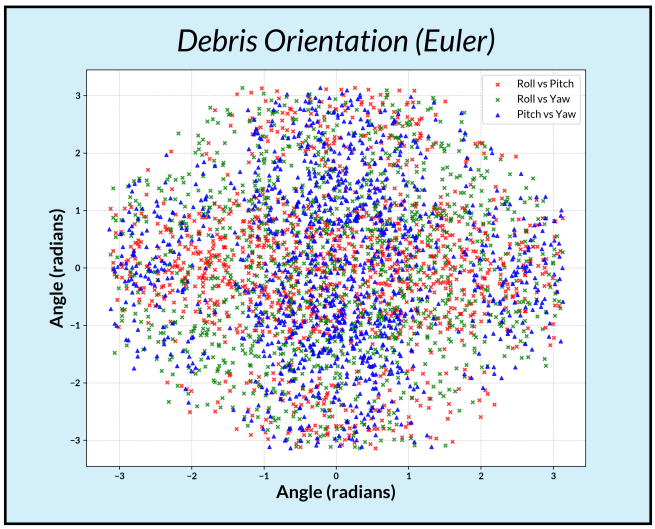


Fig. 7. Euler angle scatter plots (Roll–Pitch, Roll–Yaw, Pitch–Yaw) derived from quaternion data, showing uniformly distributed debris orientations.

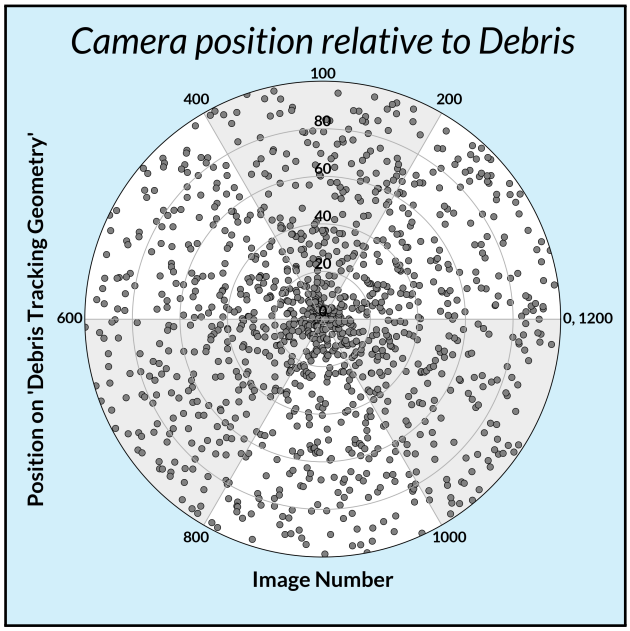


Fig. 8. Polar plot showing the camera's position relative to the debris across image captures.

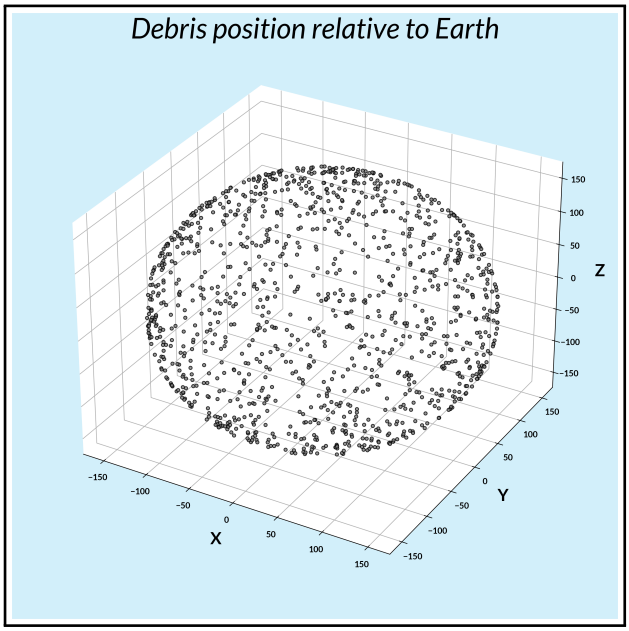


Fig. 9. 3D scatter plot showing debris positions relative to Earth.

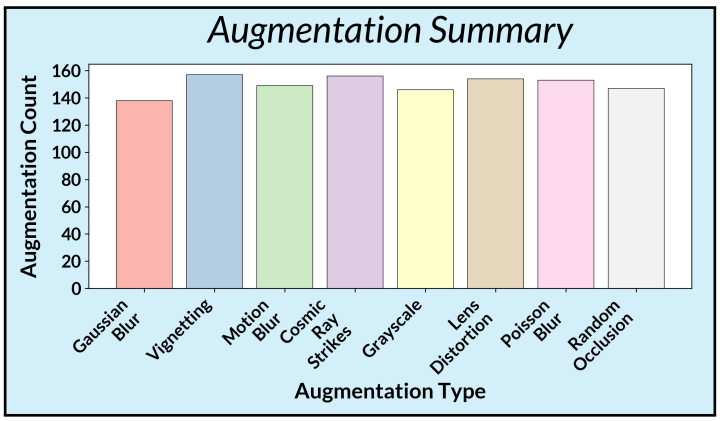


Fig. 10. Bar chart summarizing the count of each image augmentation type applied.

Fig. 7 shows scatter plots of angle pairs—Roll vs. Pitch, Roll vs. Yaw, and Pitch vs. Yaw—derived from quaternions (see Fig. C.1), which were used to represent the orientation of space debris. The data is evenly distributed across the full angular range ($-\pi$ to π) without any visible clustering. Fig. 8 shows a polar scatter plot with image number on the angular axis and camera position on `Debris_Tracking_Geometry` on the radial axis. Each of the 1,200 images was generated with a unique camera position at a fixed distance of 2 meters around the debris. The distribution of points indicates variation in camera positioning. Fig. 9 shows a 3D scatter plot of the debris positions relative to the Earth, with each point representing a unique instance from the 1,200 generated images. The debris were positioned at a fixed distance of 14 meters from the scaled Earth model, which corresponds to a LEO altitude of approximately 560 km at a 40,000:1 scale. The spatial spread of points reflects variation in debris positioning. Fig. 10 shows a bar chart with augmentation types on the x-axis and their corresponding application counts on the y-axis. Each image distortion was applied with approximately uniform frequency across the 1,200 augmented images. Following this step, the synthetic image dataset consisted of 2,400 images across four debris classes. See Figs. C.2–C.9 for examples of each image distortion.

Fig. 12 presents mAP@[50:95] values over the same training period. Both figures show that mAP values increased with training epochs. Table 3 lists the final mAP@50 and mAP@[50:95] scores at epoch 100 for all three models. The best-performing model weights, saved as `best.pt`, corresponded to the checkpoint with the highest mAP@[50:95] on the validation dataset. At the end of training, each YOLO model used its respective `best.pt` weights to run inference on the validation set once more. A confusion matrix was generated from the resulting predictions. Fig. 13a, Fig. 13b, and Fig. 13c show the confusion matrices generated by YOLOv8n, YOLOv11n, and YOLOv12n, respectively.

Table 3. Final mAP@50 and mAP@[50:95] values at epoch 100 for YOLOv8n, YOLOv11n, and YOLOv12n models.

Model	mAP@50 value at epoch 100	mAP@[50:95] value at epoch 100
YOLOv8n	0.99500	0.97674
YOLOv11n	0.99500	0.97628
YOLOv12n	0.99459	0.96937

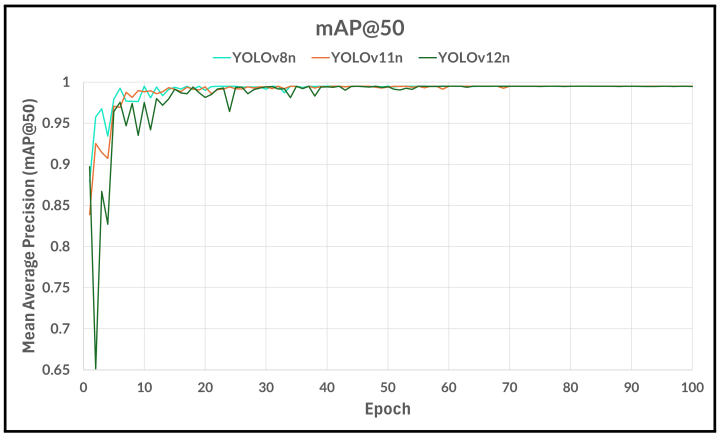


Fig. 11. Comparison of YOLOv8n, YOLOv11n, and YOLOv12n models based on mAP@50 across training epochs.

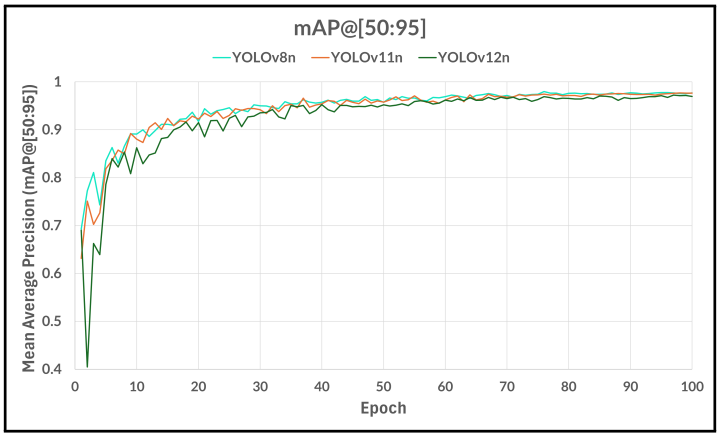
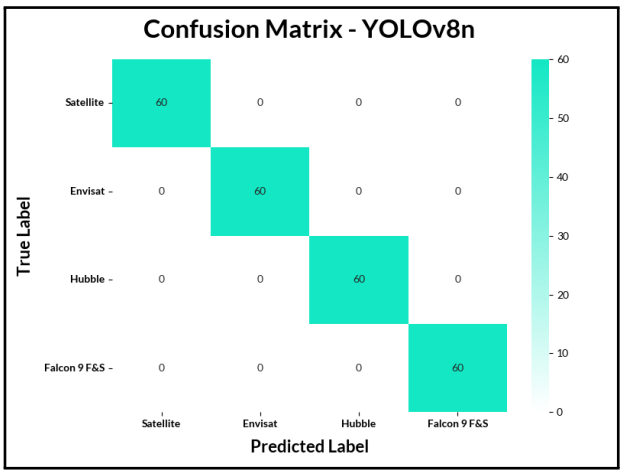
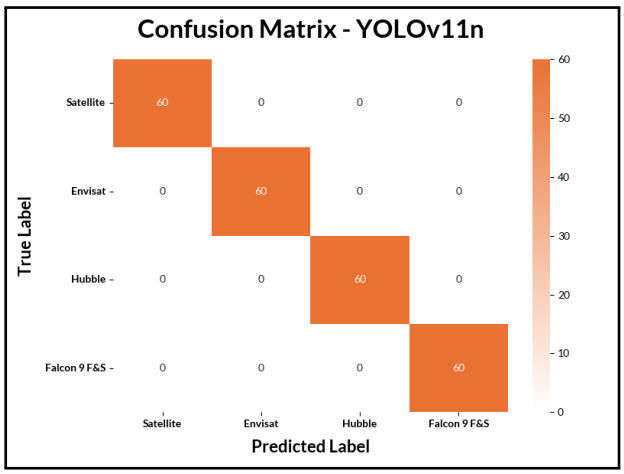


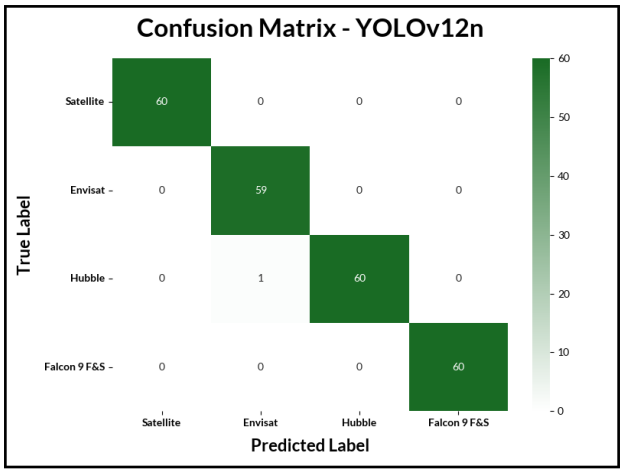
Fig. 12. Comparison of YOLOv8n, YOLOv11n, and YOLOv12n models based on mAP@[50:95] across training epochs.



(a) Confusion matrix for YOLOv8n



(b) Confusion matrix for YOLOv11n



(c) Confusion matrix for YOLOv12n

Fig. 13. Confusion matrices for 3 variants of the YOLO family of models: YOLOv8n, YOLOv11n, YOLOv12n

Once training and evaluation were completed, the best-performing model weights were used to run inference using the code in Table B.18, with a confidence threshold of 0.60. Inference was performed on two datasets that were not seen during training: the synthetic test dataset and a small set of real-world space debris images. Fig. 14 shows predicted bounding boxes with class labels and confidence scores from the YOLOv8n model on synthetic images representing three debris classes. Fig. 15 shows predictions on real-world images. Fig.16 and Fig. 17 show the corresponding results for the YOLOv11n model. Fig. 18 and Fig.19 show the corresponding results for the YOLOv12n model.

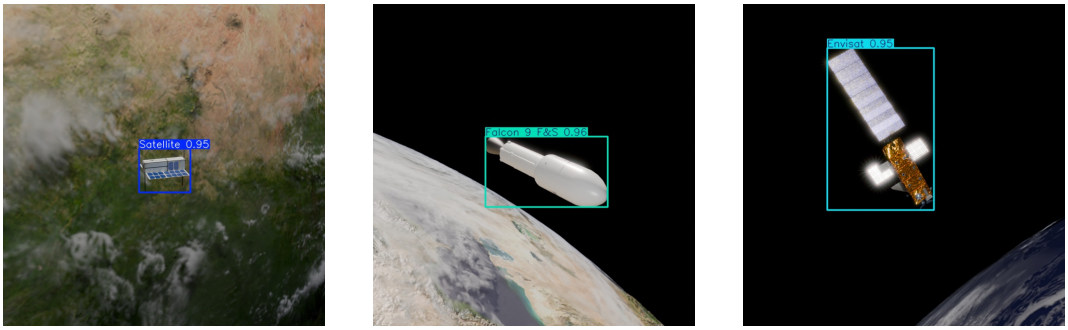


Fig. 14. YOLOv8n: Inference on unseen synthetic images

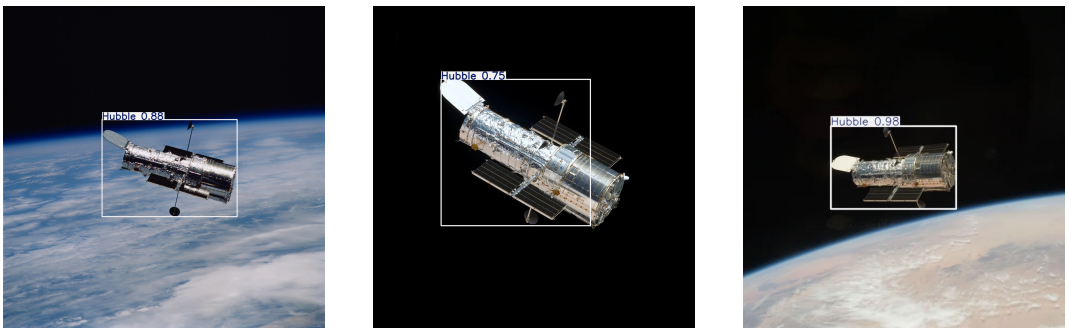


Fig. 15. YOLOv8n: Inference on real-world images

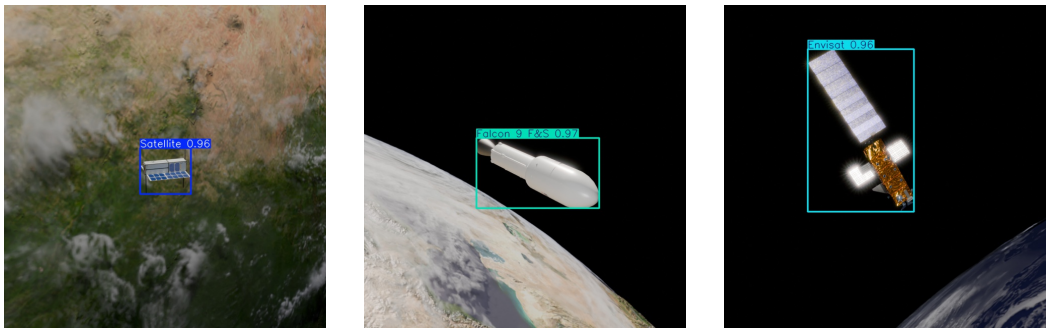


Fig. 16. YOLOv11n: Inference on unseen synthetic images

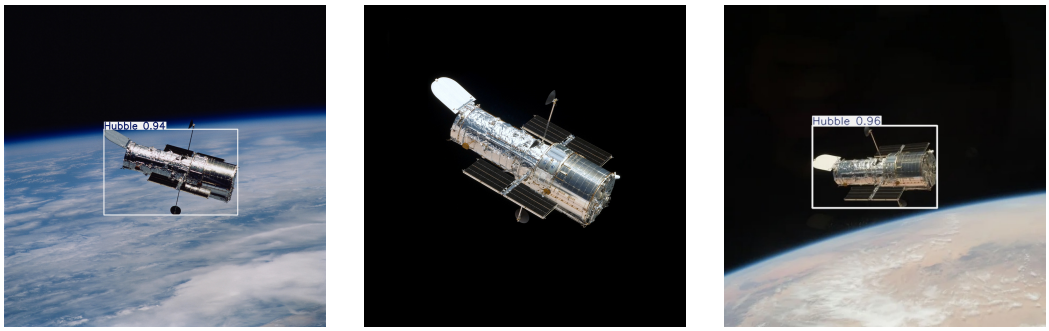


Fig. 17. YOLOv11n: Inference on real-world images

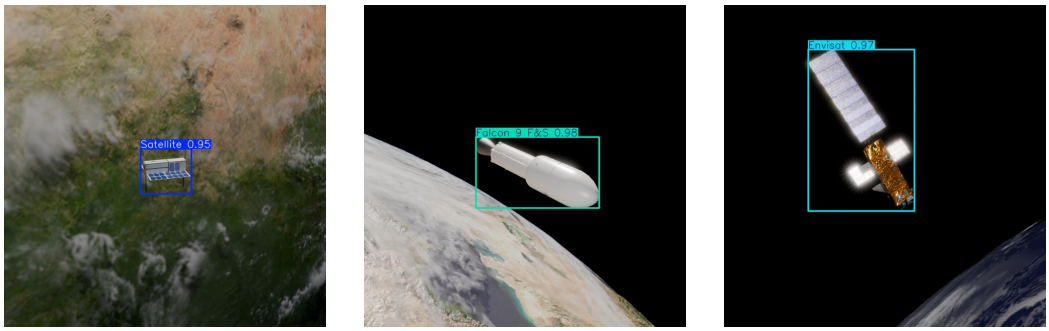


Fig. 18. YOLOv12n: Inference on unseen synthetic images

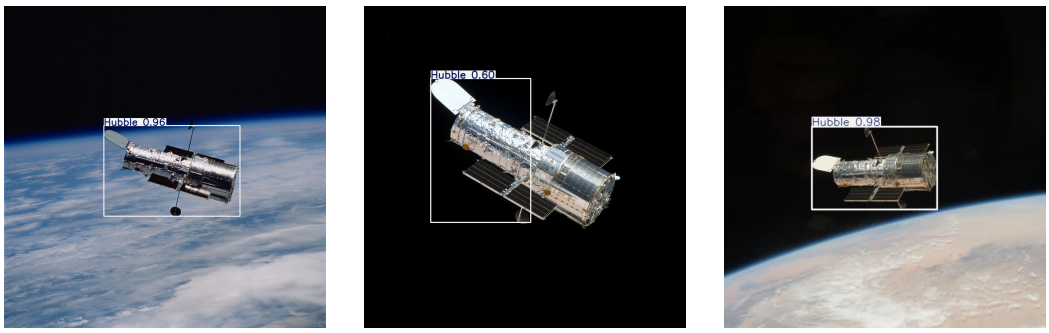


Fig. 19. YOLOv12n: Inference on real-world images

5 Discussion and future work

This study investigated the feasibility of using synthetic image datasets and lightweight DL models to enable real-time object detection of uncooperative and unpredictable space debris, particularly in the context of ADR missions. The core research questions underpinning this research were: (1) Can synthetic image datasets, generated under resource-constrained conditions, provide sufficient diversity and photorealism to train object detection models for space debris classification? (2) Can lightweight DL models, such as variants of YOLO, perform accurate object classification on both synthetic and real-world images? (3) To what extent can domain randomization and data augmentation improve model generalization to unseen, real-world images? In addressing these questions, the project aimed to reduce reliance on high-cost, hardware-intensive data pipelines and demonstrate a scalable, local, and low-cost approach to train visual navigation models for ADR missions.

The results demonstrated that domain randomization—implemented through varying debris orientation, camera positioning, debris location, cloud cover, and sun illumination—effectively diversified the synthetic image dataset across full angular and spatial ranges (see Figs.7, 8, and 9). This variation was essential to reduce overfitting, improve model generalization, and mimic the inherent randomness of debris pose and environment in orbit. Although the term "domain randomization" is not explicitly used, Armstrong et al.[15] observed that models trained on synthetic images of known spacecraft performed poorly on unfamiliar and real-world targets, attributing this to limited variability in their data generation pipeline. Their recognition of the need for more representative training data and simulation of camera effects supports the domain randomization strategy employed in this project.

Rotational randomization was implemented using quaternions to achieve uniform sampling across 3D rotational space, thereby avoiding the singularities and axis biases associated with direct Euler angle sampling. To visualize the effectiveness of this method, the quaternions were converted into Euler angles and plotted (see Fig. 7). The even distribution across all angle pairs confirms uniform coverage and the absence of gimbal lock. While the importance of diverse orientation has been acknowledged in the literature, none of the reviewed studies explicitly employed quaternion-based rotational sampling. This represents a valuable methodological contribution and suggests a potential best practice for synthetic image dataset generation.

To enhance variation beyond geometric parameters, image augmentations were applied using OpenCV [50] and Albumentations [51] libraries (Fig.10). This approach enabled fast, configurable, and randomized preprocessing. In contrast, Schubert et al.[18] integrated augmentations within `Blender`'s compositor node tree, which was computationally intensive and less flexible. By decoupling augmentation from rendering, this project reduced image generation time to an average of 14 seconds for rendering and 0.0165 seconds for augmentation—compared to the 2-minute average rendering time reported by Schubert et al.

All three YOLO models demonstrated strong performance, with YOLOv8n achieving the highest mAP@[50:95] score of 0.97674 at epoch 100 (see Table 3)—a strong indicator of consistent debris detection performance across IoU thresholds. While near-perfect mAP@50 scores were observed across models, mAP@[50:95] served as a more rigorous metric to assess bounding box quality, leading to slightly lower values.

Confusion matrices (see Figs.13a–13c) revealed minimal misclassifications, supporting the adequacy of the 2,400-image dataset in learning class-specific features. This is particularly noteworthy when compared with Armstrong et al.[15], who used nearly 50,000 images yet achieved lower detection accuracy. Compared to prior studies [15], [16], [18], [52], the pipeline developed in this project demonstrated strong classification performance using fewer, yet more strategically generated and augmented, training samples.

When evaluated on unseen synthetic and real-world images, the models performed well on debris that appeared centered and moderately sized within the frame (see Figs. 14–19). However, inference degraded when debris appeared off-center, leading to frequent FPs and lower confidence scores. This shortfall is attributed to a positional bias in the synthetic image dataset, where all debris instances were centered due to limitations in the synthetic image generation pipeline. Although this does not diminish the value of the dataset, it exposes a key weakness in generalizing to real-world scenarios, particularly where debris may appear at arbitrary positions in the camera's field of view.

Additionally, this project did not conduct quantitative validation of image photorealism, such as using shadow or feature index comparisons, due to the unavailability of high-quality real images from space agencies. In contrast, Bechini et al. [16] validated photorealism using real-world data from NASA and ESA. The real-world images used in this study were limited in quality, variety, and resolution, constraining the breadth of validation. Furthermore, the synthetic pipeline did not simulate atmospheric scattering or optical diffusion, limiting photometric realism. This affected the ability to rigorously assess the photorealistic characteristics of the synthetic image dataset.

Nonetheless, the findings suggest that with sufficient domain randomization and photorealism, lightweight YOLO models trained on procedurally generated synthetic image datasets can generalize better than expected to both unseen synthetic and real-world data. This is likely a result of accurate and consistent labeling and extensive image augmentation strategies. The synthetic image generation pipeline developed in this project contributes a low-cost, scalable method for generating photorealistic training data in scenarios where real space debris imagery is unavailable—a challenge consistently highlighted across studies [15], [16], [18] and the wider space industry.

Future work could focus on enhancing the photorealism and representativeness of the synthetic image dataset through several extensions. Implementing quantitative validation techniques—such as histogram, shadow index, and disparity map comparisons—would improve dataset evaluation, as demonstrated by Bechini et al. [16]. Incorporating camera lens models or integrating rendering frameworks like POV-Ray could enhance photorealism. Additionally, modeling sensor effects such as bloom, lens flare, or diffraction using `Blender`'s compositor could add important real-world artifacts to the synthetic image dataset.

Expanding the dataset to include RGB masks or semantic segmentation labels would also enable future experiments on pixel-level learning tasks, in line with Armstrong et al. [15]. Further, implementing position randomization to remove the central bias observed in this study would better prepare models for real-world deployment. Finally, future efforts could explore extending this pipeline to enable pose estimation and real-time onboard inference, particularly for vision-based navigation in ADR missions.

6 Conclusions

This project set out to explore whether synthetic image datasets and lightweight object detection models could enable real-time classification of uncooperative and unpredictable space debris—an essential step toward safer, more autonomous ADR missions. The end-to-end pipeline developed—from `Blender`-based image generation to YOLO training, evaluation, and inference—achieved high performance while remaining compatible with consumer-grade hardware. With a best mAP@[50:95] of 0.97674, the pipeline outperformed prior studies [18], [15], [16], [52] that relied on significantly larger datasets, owing to methodological advances such as quaternion-based rotational sampling and efficient post-render augmentation using OpenCV and Albumentations. These techniques enhanced dataset diversity, improving generalization to unseen synthetic and limited real-world images. The results validate the use of scalable, low-cost approaches to training debris detection models, showing that photorealism, diversity, and accurate labelling can substitute hard-to-obtain, expensive real-world images.

The modular design and high generalization performance suggest that the pipeline is a strong foundation for future work. As synthetic image datasets grow more photorealistic and diverse and model architectures evolve, such pipelines could be adapted for segmentation, pose estimation, and onboard inference during in-orbit servicing operations. In this way, the work contributes not just to a technical tool, but a step toward more robust and accessible solutions for navigating increasingly congested orbital environments.

References

[1] S. Sheikh. "AI has been around for decades, it's not new | LinkedIn." (), [Online]. Available: https://www.linkedin.com/pulse/ai-has-been-around-decades-its-new-saima-sheikh-uaxef/ (visited on 10/09/2024) (cited on p. 9).

[2] "AI vs. machine learning: How do they differ?" Google Cloud. (), [Online]. Available: https://cloud.google.com/learn/artificial-intelligence-vs-machine-learning (visited on 02/22/2025) (cited on p. 9).

[3] "AI in space: Exploration, research, innovation, and inclusivity." (), [Online]. Available: https://www.apu.apus.edu/area-of-study/math-and-science/resources/ai-in-space/ (visited on 02/22/2025) (cited on p. 9).

[4] OpenAI, Chatgpt, https://openai.com/chatgpt, Large language model developed by OpenAI., 2025. (visited on 02/22/2025) (cited on p. 9).

[5] "Space: The $1.8 trillion opportunity for global economic growth | McKinsey." (), [Online]. Available: https://www.mckinsey.com/industries/aerospace-and-defense/our-insights/space-the-1-point-8-trillion-dollar-opportunity-for-global-economic-growth (visited on 02/23/2025) (cited on p. 9).

[6] "Tackling the growing risks of space debris in earth orbit – UK space agency blog." (Nov. 6, 2023), [Online]. Available: https://space.blog.gov.uk/2023/11/06/tackling-the-growing-risks-of-space-debris-in-earth-orbit/ (visited on 02/23/2025) (cited on pp. 9, 10, 12).

[7] D. McKnight, M. Stevenson, C. Kunstadter, and R. Arora, "UPDATING THE MASSIVE COLLISION MONITORING ACTIVITY - CREATING a LEO COLLISION RISK CONTINUUM," (cited on p. 10).

[8] "FCC adopts new '5-year rule' for deorbiting satellites | federal communications commission." (Sep. 29, 2022), [Online]. Available: https://www.fcc.gov/document/fcc-adopts-new-5-year-rule-deorbiting-satellites (visited on 02/24/2025) (cited on p. 10).

[9] "Spacecraft to remove orbital debris | t2 portal." (), [Online]. Available: https://technology.nasa.gov/patent/MSC-TOPS-90 (visited on 12/15/2024) (cited on p. 10).

[10] G. J. J. Korf, "Pose estimation of space objects with neural networks," (cited on pp. 10, 12).

[11] "RemoveDEBRIS | space infrastructure | airbus space." (), [Online]. Available: https://www.airbus.com/en/products-services/space/earth-observation/removedebris?utm_source=chatgpt.com (visited on 02/26/2025) (cited on p. 10).

[12] "ELSA-d (end-of-life service by astroscale demonstration) - eoPortal." (), [Online]. Available: https://www.eoportal.org/satellite-missions/elsa-d (visited on 02/26/2025) (cited on p. 10).

[13] P. Jia, Q. Liu, and Y. Sun, "Detection and Classification of Astronomical Targets with Deep Neural Networks in Wide Field Small Aperture Telescopes," en, The Astronomical Journal, vol. 159, no. 5, p. 212, May 2020, arXiv:2002.09211 [astro-ph], ISSN: 0004-6256, 1538-3881. DOI: 10.3847/1538-3881/ab800a. [Online]. Available: http://arxiv.org/abs/2002.09211 (visited on 04/05/2025) (cited on pp. 12, 14).

[14] Y. Guo, X. Yin, Y. Xiao, Z. Zhao, X. Yang, and C. Dai, "Enhanced YOLOv8-based method for space debris detection using cross-scale feature fusion," en, Discover Applied Sciences, vol. 7, no. 2, p. 95, Jan. 2025, ISSN: 3004-9261. DOI: 10.1007/s42452-025-06502-7. [Online]. Available: https://link.springer.com/10.1007/s42452-025-06502-7 (visited on 04/05/2025) (cited on pp. 12, 14).

[15] W. S. Armstrong, S. Drakontaidis, and N. Lui, 10, Nov. 22, 2022. DOI: 10.48550/arXiv.2211.11941. arXiv: 2211.11941[cs]. [Online]. Available: http://arxiv.org/abs/2211.11941 (visited on 03/02/2025) (cited on pp. 13, 14, 37–39).

[16] M. Bechini, M. Lavagna, and P. Lunghi, "4," Acta Astronautica, vol. 204, pp. 358–369, Mar. 2023, ISSN: 00945765. DOI: 10.1016/j.actaastro.2023.01.012. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0094576523000127 (visited on 03/03/2025) (cited on pp. 13, 38, 39).

[17] T. H. Park, M. Märtens, G. Lecuyer, D. Izzo, and S. D'Amico, "SPEED+: Next-Generation Dataset for Spacecraft Pose Estimation across Domain Gap," en, in 2022 IEEE Aerospace Conference (AERO), arXiv:2110.03101 [cs], Mar. 2022, pp. 1–15. DOI: 10.1109/AERO53065.2022.9843439. [Online]. Available: http://arxiv.org/abs/2110.03101 (visited on 04/05/2025) (cited on pp. 13, 14).

[18] C. Schubert, K. Black, D. Fonseka, et al., "1," 2021 IEEE Aerospace Conference (50100), pp. 1–15, Mar. 2021, Conference Name: 2021 IEEE Aerospace Conference ISBN: 9781728174365 Place: Big Sky, MT, USA Publisher: IEEE. DOI: 10.1109/AERO50100.2021.9438232. [Online]. Available: https://ieeexplore.ieee.org/document/9438232/ (visited on 02/28/2025) (cited on pp. 13, 16, 38, 39).

[19] M. Pugliatti and F. Topputo, Boulders Identification on Small Bodies Under Varying Illumination Conditions, en, arXiv:2210.16283 [cs], Oct. 2022. DOI: 10.48550/arXiv.2210.16283. [Online]. Available: http://arxiv.org/abs/2210.16283 (visited on 04/05/2025) (cited on pp. 13, 14).

[20] J. Allworth, L. Windrim, J. Wardman, D. Kucharski, J. Bennett, and M. Bryson, Development of a High Fidelity Simulator for Generalised Photometric Based Space Object Classification using Machine Learning, en, arXiv:2004.12270 [physics], Apr. 2020. DOI: 10.48550/arXiv.2004.12270. [Online]. Available: http://arxiv.org/abs/2004.12270 (visited on 04/05/2025) (cited on p. 13).

[21] M. Nussbaum, E. Schafer, Z. Yoon, D. Keil, and E. Stoll, "Spectral Light Curve Simulation for Parameter Estimation from Space Debris," en, Aerospace, vol. 9, no. 8, p. 403, Jul. 2022, ISSN: 2226-4310. DOI: 10.3390/aerospace9080403. [Online]. Available: https://www.mdpi.com/2226-4310/9/8/403 (visited on 04/05/2025) (cited on p. 13).

[22] R. Clark, Y. Fu, S. Dave, and R. Lee, "Simulation of RSO Images for Space Situation Awareness (SSA) Using Parallel Processing," en, Sensors, vol. 21, no. 23, p. 7868, Nov. 2021, ISSN: 1424-8220. DOI: 10.3390/s21237868. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7868 (visited on 04/05/2025) (cited on p. 13).

[23] H. Zeng and Y. Xia, "Space target recognition based on deep learning," en, in 2017 20th International Conference on Information Fusion (Fusion), Xi'an, China: IEEE, Jul. 2017, pp. 1–5, ISBN: 978-0-9964527-0-0. DOI: 10.23919/ICIF.2017.8009786. [Online]. Available: http://ieeexplore.ieee.org/document/8009786/ (visited on 04/05/2025) (cited on p. 14).

[24] J. Xi, Y. Xiang, O. K. Ersoy, M. Cong, X. Wei, and J. Gu, "Space debris detection using feature learning of candidate regions in optical image sequences," IEEE Access, vol. 8, pp. 150 864–150 877, 2020. DOI: 10.1109/ACCESS.2020.3016761 (cited on p. 14).

[25] B. Foundation, Rendering, en. [Online]. Available: https://www.blender.org/features/rendering/ (visited on 03/06/2025) (cited on p. 16).

[26] Admin, Astroscale Secures Series D Round Funding to Clean Low Earth Orbit, Bringing Total Capital Raised to US $102 Million, en, Oct. 2018. [Online]. Available: https://astroscale.com/astroscale-secures-series-d-round-funding-to-clean-low-earth-orbit-bringing-total-capital-raised-to-us-102-million/ (visited on 03/06/2025) (cited on p. 16).

[27] F. Massimi, P. Ferrara, R. Petrucci, and F. Benedetto, "Deep learning-based space debris detection for space situational awareness: A feasibility study applied to the radar processing," IET Radar, Sonar & Navigation, vol. 18, no. 4, pp. 635–648, 2024, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/rsn2.12547, ISSN: 1751-8792. DOI: 10.1049/rsn2.12547. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1049/rsn2.12547 (visited on 03/07/2025) (cited on p. 17).

[28] "YOLOv8 vs faster r-CNN: A comparative analysis | keylabs," Keylabs: latest news and updates. (Jan. 15, 2024), [Online]. Available: https://keylabs.ai/blog/yolov8-vs-faster-r-cnn-a-comparative-analysis/ (visited on 03/07/2025) (cited on p. 17).

[29] OpenMMLab. "Dive into YOLOv8: How does this state-of-the-art model work?" Medium. (Jan. 17, 2023), [Online]. Available: https://openmmlab.medium.com/dive-into-yolov8-how-does-this-state-of-the-art-model-work-10f18f74bab1 (visited on 03/07/2025) (cited on p. 18).

[30] "Technology | 2024 stack overflow developer survey." (), [Online]. Available: https://survey.stackoverflow.co/2024/technology/#most-popular-technologies (visited on 03/08/2025) (cited on p. 18).

[31] "Anaconda | understanding conda and pip," Anaconda. (Nov. 28, 2018), [Online]. Available: https://www.anaconda.com/blog/understanding-conda-and-pip (visited on 03/08/2025) (cited on p. 19).

[32] B. Pryke, How to Use Jupyter Notebook: A Beginner's Tutorial, en-US, Jan. 2025. [Online]. Available: https://www.dataquest.io/blog/jupyter-notebook-tutorial/ (visited on 03/09/2025) (cited on p. 19).

[33] What is Data Augmentation? - Data Augmentation Techniques Explained - AWS, en-US. [Online]. Available: https://aws.amazon.com/what-is/data-augmentation/ (visited on 03/09/2025) (cited on p. 19).

[34] A Complete Guide to Data Augmentation, en. [Online]. Available: https://www.datacamp.com/tutorial/complete-guide-data-augmentation (visited on 03/09/2025) (cited on p. 19).

[35] S. Samana. "ML model evaluation: Ensuring reliability and performance in production," Pecan AI. (Aug. 23, 2024), [Online]. Available: https://www.pecan.ai/blog/ml-model-evaluation-reliability-performance/ (visited on 03/11/2025) (cited on p. 20).

[36] A. Tiwari, "Chapter 2 - supervised learning: From theory to applications," in Artificial Intelligence and Machine Learning for EDGE Computing, R. Pandey, S. K. Khatri, N. k. Singh, and P. Verma, Eds., Academic Press, Jan. 1, 2022, pp. 23–32, ISBN: 978-0-12-824054-0. DOI: 10.1016/B978-0-12-824054-0.00026-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128240540000265 (visited on 03/11/2025) (cited on p. 22).

[37] "Mean average precision (mAP) explained: Everything you need to know." (), [Online]. Available: https://www.v7labs.com/blog/mean-average-precision (visited on 03/12/2025) (cited on p. 23).

[38] A. Anwar. "What is average precision in object detection & localization algorithms and how to calculate it?" TDS Archive. (May 13, 2022), [Online]. Available: https://medium.com/data-science/what-is-average-precision-in-object-detection-

localization-algorithms-and-how-to-calculate-it-3f330efe697b (visited on 03/14/2025) (cited on p. 23).

[39] Planet Earth Texture Maps. [Online]. Available: https://planetpixelemporium.com/earth.html (visited on 03/15/2025) (cited on pp. 24, 27).

[40] Folder - Google Drive. [Online]. Available: https://drive.google.com/drive/folders/1MRwpedip8EJVlm7YZi7lvEajPPN6lBwj (visited on 03/15/2025) (cited on p. 24).

[41] Blender Guru, Blender Tutorial: Realistic Earth, Aug. 2022. [Online]. Available: https://www.youtube.com/watch?v=0YZzHn0iz8U (visited on 03/15/2025) (cited on p. 26).

[42] UV Issues following Blender Guru tut 'Realistic Earth' - Support / Materials and Textures, en, Section: Support, Sep. 2022. [Online]. Available: https://blenderartists.org/t/uv-issues-following-blender-guru-tut-realistic-earth/1404800 (visited on 03/15/2025) (cited on p. 27).

[43] Space Debris - NASA, en-US, Section: NASA Headquarters. [Online]. Available: https://www.nasa.gov/headquarters/library/find/bibliographies/space-debris/ (visited on 03/16/2025) (cited on p. 27).

[44] Space debris by the numbers, en. [Online]. Available: https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers (visited on 03/16/2025) (cited on p. 27).

[45] "Jaikr-dev/space-debris-detection-pipeline," GitHub. (), [Online]. Available: https://github.com/jaikr-dev/space-debris-detection-pipeline (visited on 04/10/2025) (cited on pp. 28–31, 46–51).

[46] Space debris by the numbers, en. [Online]. Available: https://www.esa.int/Space_Safety/Space_Debris/Space_debris_by_the_numbers (visited on 03/16/2025) (cited on p. 28).

[47] information@eso.org, Soft Capture Mechanism, en. [Online]. Available: https://esahubble.org/about/general/soft_capture/ (visited on 03/18/2025) (cited on p. 28).

[48] SpaceX launches 21 Starlink satellites on Falcon 9 rocket from Cape Canaveral – Spaceflight Now, en-US. [Online]. Available: https://spaceflightnow.com/2025/01/27/live-coverage-spacex-to-launch-21-starlink-satellites-on-falcon-9-rocket-from-cape-canaveral-7/ (visited on 03/17/2025) (cited on p. 28).

[49] geoffc, Answer to "What happens to the Falcon 9 second stage after payload separation?", Jan. 2015. [Online]. Available: https://space.stackexchange.com/a/7815 (visited on 03/17/2025) (cited on p. 28).

[50] "Home," OpenCV. (), [Online]. Available: https://opencv.org/ (visited on 03/31/2025) (cited on p. 38).

[51] "Albumentations: Fast and flexible image augmentations," Albumentations. (), [Online]. Available: https://albumentations.ai/ (visited on 03/31/2025) (cited on p. 38).

[52] Y. Guo, X. Yin, Y. Xiao, Z. Zhao, X. Yang, and C. Dai, "Enhanced YOLOv8-based method for space debris detection using cross-scale feature fusion," Discover Applied Sciences, vol. 7, no. 2, p. 95, Jan. 21, 2025, ISSN: 3004-9261. DOI: 10.1007/s42452-025-06502-7. [Online]. Available: https://link.springer.com/10.1007/s42452-025-06502-7 (visited on 03/31/2025) (cited on pp. 38, 39).

[53] S. Sheikh, AI has been around for decades, it's not new | LinkedIn. [Online]. Available: https://www.linkedin.com/pulse/ai-has-been-around-decades-its-new-saima-sheikh-uaxef/ (visited on 10/09/2024) (cited on p. 61).

[54] B. Marr, Artificial Intelligence In Space: The Amazing Ways Machine Learning Is Helping To Unravel The Mysteries Of The Universe, en, Section: Enterprise Tech. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2023/04/10/artificial-intelligence-in-space-the-amazing-ways-machine-learning-is-helping-to-unravel-the-mysteries-of-the-universe/ (visited on 10/09/2024) (cited on p. 61).

[55] What Is Edge AI? Benefits and Use Cases, en. [Online]. Available: https://www.run.ai/guides/machine-learning-operations/edge-ai (visited on 10/05/2024) (cited on p. 61).

[56] Tackling the growing risks of space debris in Earth orbit – UK Space Agency blog, en, Nov. 2023. [Online]. Available: https://space.blog.gov.uk/2023/11/06/tackling-the-growing-risks-of-space-debris-in-earth-orbit/ (visited on 10/09/2024) (cited on pp. 61, 62).

A Blender Setup

Table A.1. Scene Primary Axis setup

Scene Primary Axis: Serves as a placeholder for the Earth in the script.
Add an Empty Object: Add→Empty→Plain Axis • The object is named `Earth_Axis` in the `.blend` file, as referenced in [45]. • Scale: `X = Y = Z = 200`
Align origin to `Surface` origin: • Select `Earth_Axis`, then (Object→Set Origin→Origin to Geometry). • Select `Surface`, then `Shift + S`→2 (Cursor to Selected). • Select `Earth_Axis`, then `Shift + S`→8 (Selection to Cursor).

Table A.2. Orbit Geometry setup

Orbit Geometry: Defines the orbital path for space debris.
Add a Circular Curve: Add→Curve→Bezier Circle • The object is named `Orbit_Geometry` in the `.blend` file, as referenced in [45]. • Dimensions: `X = Y = 349 m, Z = 0 m`
Align origin to `Surface` origin: • Select `Orbit_Geometry`, then (Object→Set Origin→Origin to Geometry). • Select `Surface`, then `Shift + S`→2 (Cursor to Selected). • Select `Orbit_Geometry`, then `Shift + S`→8 (Selection to Cursor).
Add Constraints: • In `Properties`→`Constraints`, add a `Child Of` constraint. • Set `Target` to `Earth_Axis`.

Table A.3. Orbit Container setup

Orbit Container: Facilitates position randomization of space debris.
Add an Empty Cube: Add→Empty→Cube • The object is named `Orbit_Container` in the `.blend` file, as referenced in [45]. • Scale: `X = Y = Z = 0.050`
Add Constraints: • Select `Orbit_Container`. • In `Properties`→`Constraints`, add a `Follow Path` constraint. • Set `Target` to `Orbit_Geometry`.

Table A.4. Orbit Container Axis Setup

Orbit Container Axis: Facilitates orientation randomization of space debris.
Add an Empty Object: Add→Empty→Plain Axis • The object is named `Orbit_Container_Axis` in the `.blend` file, as referenced in [45]. • Scale: `X = Y = Z = 10`
Align origin to `Orbit_Container` origin: • Select `Orbit_Container_Axis`, then (Object→Set Origin→Origin to Geometry). • Select `Orbit_Container`, then `Shift + S`→2 (Cursor to Selected). • Select `Orbit_Container_Axis`, then `Shift + S`→8 (Selection to Cursor).
Add Constraints: • Select `Orbit_Container_Axis`. • In `Properties`→`Constraints`, add a `Child Of` constraint. • Set `Target` to `Orbit_Container`.

Table A.5. Debris Tracking Geometry setup

Debris Tracking Geometry: Defines a circular path for `Debris_Tracking_Container`.
Add a Bezier Circle: Add→Curve→Bezier Circle • The object is named `Debris_Tracking_Geometry` in the `.blend` file, as referenced in [45]. • Dimensions: `X = Y = 4 m, Z = 0`
Align origin to `Orbit_Container_Axis` origin: • Select `Debris_Tracking_Geometry`, then (Object→Set Origin→Origin to Geometry). • Select `Orbit_Container_Axis`, then `Shift + S`→`2` (Cursor to Selected). • Select `Debris_Tracking_Geometry`, then `Shift + S`→`8` (Selection to Cursor).
Add Constraints: • Select `Debris_Tracking_Geometry`. • In `Properties`→`Constraints`, add a `Child Of` constraint. • Set `Target` to `Orbit_Container_Axis`.

Table A.6. Debris Tracking Container setup

Debris Tracking Container: Facilitates position randomization of `Camera`.
Add an Empty Cube: Add→Empty→Cube • The object is named `Debris_Tracking_Container` in the `.blend` file, as referenced in [45]. • Scale: `X = Y = Z = 0.050`
Add Constraints: • Select `Debris_Tracking_Container`. • In `Properties`→`Constraints`, add a `Follow Path` constraint. • Set `Target` to `Debris_Tracking_Geometry`.

Table A.7. Camera Geometry setup

Camera Geometry: Defines the circular path for `Camera`.
Add a Bezier Circle: Add→Curve→Bezier Circle • The object is named `Camera_Geometry` in the `.blend` file, as referenced in [45]. • Dimensions: `X = Z = 0.5 m, Y = 0`
Align origin to `Debris_Tracking_Container` origin: • Select `Camera_Geometry`, then (Object→Set Origin→Origin to Geometry). • Select `Debris_Tracking_Container`, then `Shift + S`→2 (Cursor to Selected). • Select `Camera_Geometry`, then `Shift + S`→8 (Selection to Cursor).
Add Constraints: • Select `Camera_Geometry`. • In `Properties`→`Constraints`, add a `Child Of` constraint. • Set `Target` to `Debris_Tracking_Container`.
Ensure `Camera_Geometry` is perpendicular to `Debris_Tracking_Geometry`.

Table A.8. Camera Container setup

Camera Container: Serves as a placeholder for `Camera` in the script.
Add an Empty Cube: Add→Empty→Cube • The object is named `Camera_Container` in the `.blend` file, as referenced in [45]. • Scale: `X = Y = Z = 0.050`
Add Constraints: • Select `Camera_Container`. • In `Properties`→`Constraints`, add a `Follow Path` constraint. • Set `Target` to `Camera_Geometry`.

Table A.9. Camera setup

Camera: Defines the camera setup to track the debris.
Add a Camera: Add→Camera • The object is named `Camera` in the `.blend` file, as referenced in [45].
Align origin to `Camera_Container` origin: • Select `Camera`, then (Object→Set Origin→Origin to Geometry). • Select `Camera_Container`, then `Shift + S`→2 (Cursor to Selected). • Select `Camera`, then `Shift + S`→8 (Selection to Cursor).
Add Constraints: • Select `Camera`. • In `Properties`→`Constraints`, add a `Track To` constraint, and set `Target` to `Orbit_Container`. • In `Properties`→`Constraints`, add a `Child Of` constraint, and set `Target` to `Camera_Container`.
Adjust `Camera_Container`'s rotation to ensure it tracks the space debris.

Table A.10. Debris Illumination setup

Debris Illumination: Provides realistic lighting for space debris.
Add an Area Light: Add→Area→Light • The object is named `Chaser_Illumination` in the `.blend` file, as referenced in [45].
Align origin to `Debris_Tracking_Container` origin: • Select `Chaser_Illumination`, then (Object→Set Origin→Origin to Geometry). • Select `Debris_Tracking_Container`, then `Shift + S`→`2` (Cursor to Selected). • Select `Chaser_Illumination`, then `Shift + S`→`8` (Selection to Cursor).
Add Constraints: • Select `Chaser_Illumination`. • In `Properties`→`Constraints`, add a `Track To` constraint, and set `Target` to `Orbit_Container`. • In `Properties`→`Constraints`, add a `Child Of` constraint, and set `Target` to `Debris_Tracking_Container`. • Select the `Data` tab in the `Properties` panel. • Change the `Power` to `20 W` and `Size` to `10 m`.
The `Power` and `Size` values were arbitrarily set to ensure that the scene is realistically lit upon rendering.

Table A.11. Converting the Envisat.glb file into a single object

Converting the Envisat.glb file into a single object.
Open the .glb file in Blender: • Launch a separate Blender window and load the Envisat.glb file.
Select the entire model: • In the Outliner, locate and right-click on the Envisat.glb Collection, then choose Select Hierarchy to highlight the complete model.
Remove parent-child links: • Move the cursor to the 3D Viewport, press Alt + P, then select Clear and Keep Transformation.
Remove Instance objects: • In the Outliner, identify, select, and delete all Instance objects (denoted by the boomerang icon).
Merge and export the remaining objects: • In the Outliner, select all remaining objects (denoted by the triangle icon). • Move the cursor to the 3D Viewport, then press Ctrl + J to merge them into a single object. • Export the newly merged object as Envisat.glb to preserve the original material and textures.

Table A.12. Integrating the modified `Envisat.glb` file into the `Blender` scene

Integrating the modified `Envisat.glb` file into the `Blender` scene.
Import the modified `Envisat.glb` file: • Open the (Earth + Sun + LEO Geometry) `.blend` file, then import the modified `Envisat.glb` file.
Align origin to `Orbit_Container` origin: • Select `Envisat`, then (Object→Set Origin→Origin to Geometry). • Select `Orbit_Container`, then `Shift + S`→2 (Cursor to Selected). • Select `Envisat`, then `Shift + S`→8 (Selection to Cursor).
Scale the `Envisat` object: • Adjust the scale of the `Envisat` object to ensure a realistic appearance and proper fit within `Camera`'s frame.
Add Constraints: • Select `Envisat`. • In `Properties`→`Constraints`, add a `Child Of` constraint, and set `Target` to `Orbit_Container`.

B Pesudocode

Table B.13. Pseudocode for monitoring and restarting `Blender`

FUNCTION `restart_blender(delay)`
PRINT `"Starting Blender monitoring..."`
LOOP FOREVER
LAUNCH Blender in background with the specified `.blend` file and Python script
IF the process completes with an exit code: **PRINT** the `exit code`
IF the `'stop_flag.txt'` file exists: **PRINT** `"Limit reached. Stopping..."` **BREAK** from loop
WAIT `delay` seconds
PRINT `"Restarting Blender..."`

Table B.14. Pseudocode for resizing and scaling the synthetic image dataset

FUNCTION `resize_and_scale_dataset()`
FOR each split in `["train", "val", "test"]`
CREATE scaled output folders for images and labels if they don't exist
FOR each image file in the original images folder of this split
READ the image
RESIZE the image to `NEW_SIZE`
CONVERT from PNG to JPG
SAVE resized image in scaled output folder
FIND corresponding label file
IF it exists, **COPY** label to the scaled label folder (no modification needed)
CREATE a `'config_scaled.yaml'` file in the scaled dataset folder
WRITE the required structure (`train`, `val`, `test` paths, number of classes, class names)
PRINT `"Config file created."`

Table B.15. Pseudocode for visualizing bounding boxes on images

FUNCTION `visualize_bounding_boxes()`
FOR each split in `["train", "val", "test"]`
CREATE an `"annotated"` folder for that split
FOR each image in the split's images folder
READ the image
DETERMINE label file path from the image filename
IF the label file exists:
FOR each line in the label file
PARSE bounding box coordinates (`class`, `x_center`, `y_center`, `width`, `height`)
CONVERT normalized coordinates to absolute pixel values
DRAW bounding box on the image
DRAW center marker on the image
SAVE the annotated image in the `"annotated"` folder

Table B.16. Pseudocode for applying augmentations to dataset images

FUNCTION `apply_augmentations()`
`augmentation_list` = [`GaussianNoise`, `RandomOcclusion`, `MotionBlur`, `Grayscale`, ...]
CREATE a dictionary to count how many times each augmentation is applied
FOR each split in `["train", "val", "test"]`
FOR each image file in the scaled images folder for this split
SELECT one random augmentation from the list
LOAD the image (in the way that augmentation requires)
APPLY the selected augmentation
SAVE the augmented image with a new name indicating augmentation
COPY the corresponding label file (if it exists) to match the new name
INCREMENT the counter for the chosen augmentation
PRINT `"Augmentation Summary"` with all counters

Table B.17. Pseudocode for training a YOLOv8 model and logging results to W&B

FUNCTION `train_and_log_model(epochs, batch_size)`
CONFIGURE W&B with your project name and API key
CHECK GPU availability: **IF** GPU is available, note how many **ELSE** use CPU
LOAD the YOLOv8 model (e.g., `"yolov8s.pt"`)
GENERATE a unique run name (with timestamp)
START a W&B run with this name
PRINT `"Starting training with epochs, batch=batch_size"`

`model.train(`
`data="config_scaled.yaml",`
`epochs=epochs,`
`batch_size=batch_size,`
`device=(GPU index if available)`
`)`

`run_dir = "directory where training results are saved"`
`model.val(...)`
EXTRACT confusion matrix and **SAVE** to CSV
GENERATE confusion matrix plot
LOG confusion matrix CSV and plot to W&B
LOG existing results (e.g., `results.csv`, `best.pt`, training plot) to W&B
FINISH the W&B run
PRINT `"Training and logging completed."`

Table B.18. Pseudocode for running inference using a trained model

FUNCTION `run_inference(optional_model_weight_path, confidence_threshold)`
CREATE a timestamped directory for inference results
DETERMINE which model weights to use: **IF** user did not specify, use the default `best.pt` from training
PRINT `"Loading model..."`
RUN inference on all test images with the given confidence threshold
SAVE the predictions (with bounding boxes) in the inference directory
PRINT `"Inference completed. Results in <output folder>."`

Table B.19. Pseudocode for loading coordinates from a folder of text files

FUNCTION `load_coordinates(folder)`
FOR each `.txt` file in `folder`
READ the file content
PARSE the last three values as `x`, `y`, `z`
STORE them in a list
WRITE the collected `x`, `y`, `z` to a CSV
RETURN the lists of `x`, `y`, `z`

Table B.20. Pseudocode for loading and plotting 3D data and offsets from sensor files

FUNCTION `load_quaternions(folder)`
FOR each `.txt` file in `folder`
READ the file content
PARSE the last four values as `w`, `x`, `y`, `z`
STORE them
WRITE them to CSV
RETURN the lists of `w`, `y`, `x`, `z`
FUNCTION `load_offset_values(folder)`
FOR each `.txt` file in `folder`
EXTRACT integer frame number from filename
READ the content as a single offset value
STORE them keyed by frame number
SORT by frame number
WRITE frame-offset pairs to CSV
RETURN the lists of `frames` and `offsets`
FUNCTION `plot_3d_scatter(x, y, z)`
CREATE a 3D plot with the given points
FUNCTION `plot_3d_quaternion_scatter(w, x, y, z)`
CREATE a 3D plot with the given points
FUNCTION `plot_euler_angle_scatter(R, P, Y)`
CREATE a scatter plot with the given points (`R vs P`, `R vs Y`, `P vs Y`)
FUNCTION `plot_offset_values(frames, offsets)`
CREATE a polar or line plot (depending on preference)
MAP frames to angles or x-axis
PLOT offsets radially or on the y-axis
FUNCTION `main()`
`x, y, z = load_coordinates(debris_folder)` `plot_3d_scatter(x, y, z)`
`w, xQ, yQ, zQ = load_quaternions(quaternion_folder)` `plot_3d_quaternion_scatter(w, x, y, z)`
`R, P, Y = load_coordinates(euler_folder)` `plot_euler_angle_scatter(R, P, Y)`
`frames, offsets = load_offset_values(container_folder)` `plot_offset_values(frames, offsets)`

C Supporting images

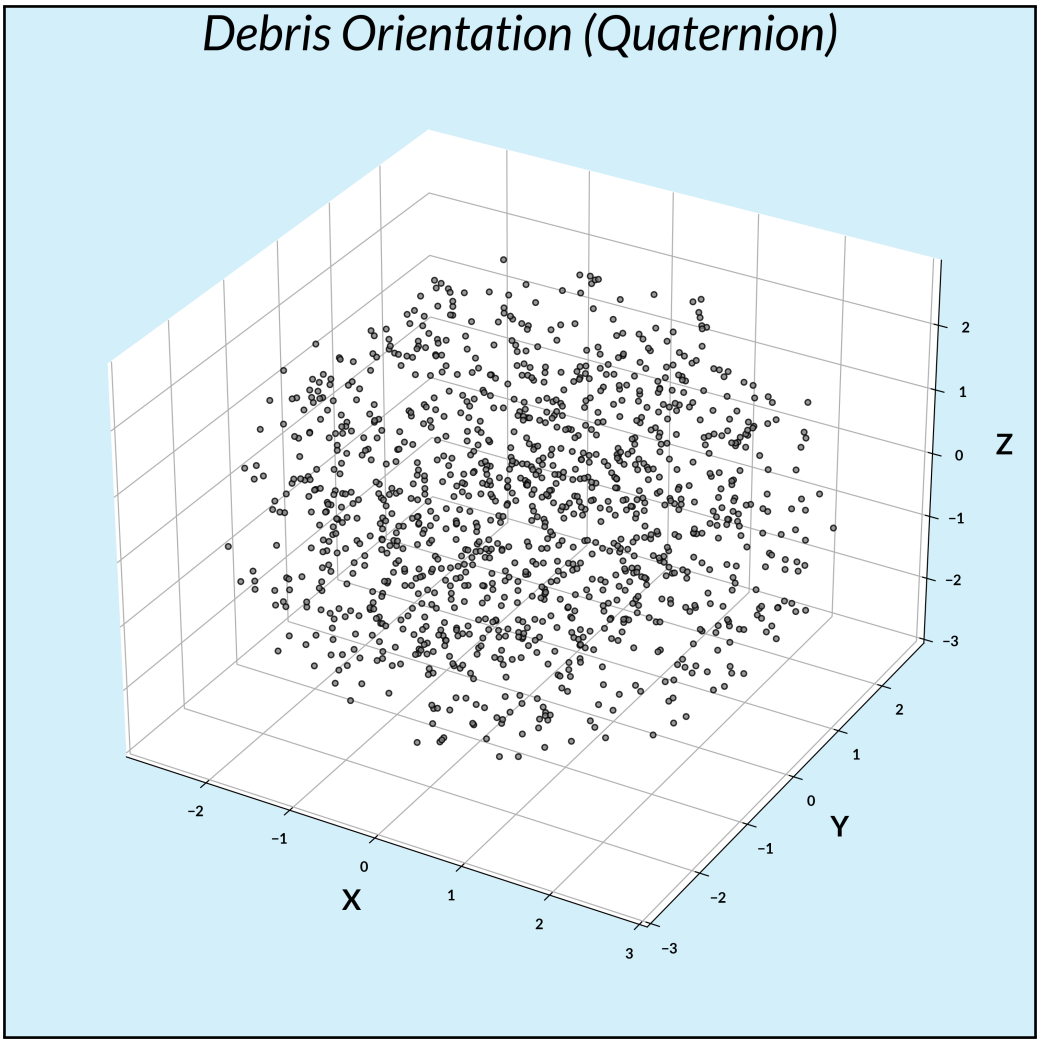


Fig. C.1. 3D scatter plot of debris orientations represented as quaternions.

Fig. C.2. Gaussian blur example

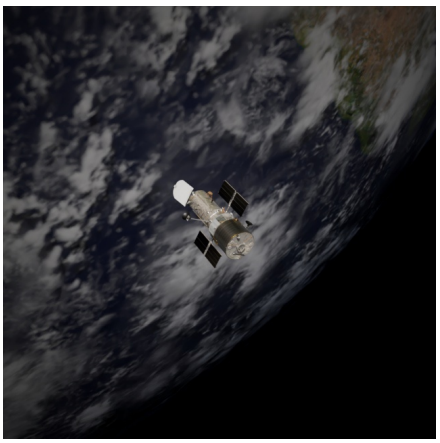


Fig. C.3. Vignette example

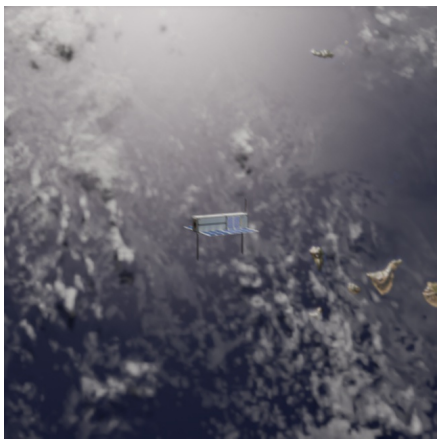


Fig. C.4. Motion blur example

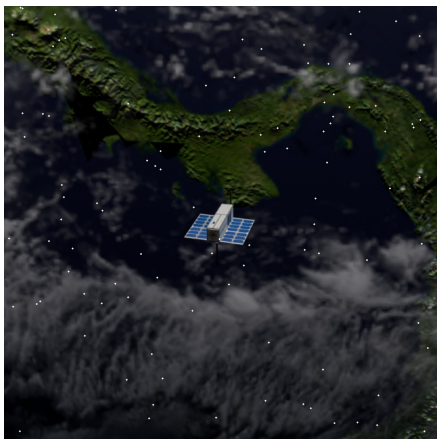


Fig. C.5. Cosmic ray strikes example

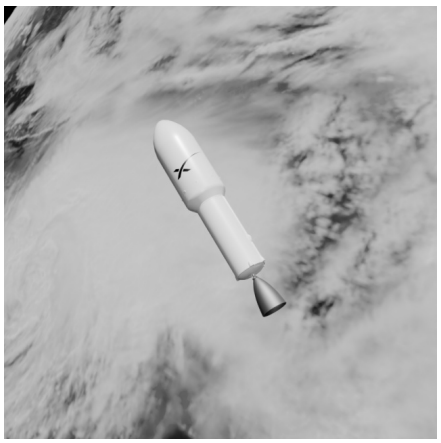


Fig. C.6. Grayscale example

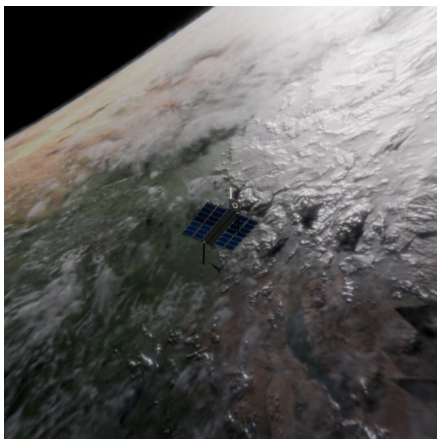


Fig. C.7. Lens distortion example

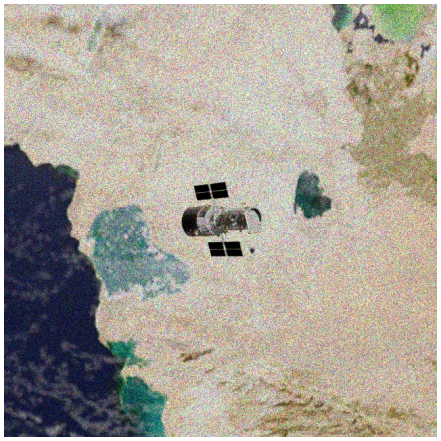


Fig. C.8. Poisson blur example

Fig. C.9. Random occlusion example

D Inital Project Outline (Deliverable 1)

D.1 Background

Artificial intelligence (AI) has been around for decades; its history dates back to the mid-20^{th} century when Alan Turing introduced the Turing Test, a method to determine whether a machine can exhibit intelligent behavior [53]. The term "Artificial Intelligence" was first coined at the Dartmouth Conference in 1956 [53]. Since then, each decade of AI development has built upon the last "showcasing human ingenuity and a relentless drive towards progress"[53]. Today, we are riding the wave of Generative AI which has reached new horizons. As you read this sentence, AI algorithms are painting cosmic portraits, composing symphonies, diagnosing medical conditions, translating complex documents, and simulating entire ecosystems. They are designing video game models, creating synthetic avatars, debugging code, and offering personalized financial planning. In short, AI has already had a permeating impact on our lives. In recent months, generative AI tools have been ravenously embraced by an intrigued and astonished public, fueled by programs like ChatGPT. As we enter the era of the fifth industrial revolution, AI is the talk of the town, but space exploration, though booming, has remained somewhat in the background. The space sector is undergoing a rapid and transformative change. Advanced propulsion systems, reusable launch vehicles, space tourism and exploration, space mining, and space real-estate are no longer futuristic concepts but emerging realities that are reshaping our future beyond Earth. However, space is vast, lonely, and hostile and "involves some of the most complex and dangerous scientific and technical operations ever carried out"[54]. Consequently, it often presents the kinds of challenges that AI is proving itself to be remarkably helpful with. AI can be leveraged for autonomous spacecraft navigation, satellite image analysis, terrain mapping for planetary exploration, autonomous Earth observation and disaster response, and space debris tracking, to name a few.

D.2 Motivation

Machine learning (ML) has grown tremendously in recent years and has made a significant impact in almost every industry. However, most state-of-the-art models are currently hosted in cloud-based centers because of their substantial computing requirements. This reliance on centralized computing limits ML from reaching its full potential. If ML were embedded in every device around and above us, our environments could evolve into seamless, intelligent ecosystems, ushering in a new era of hyper-connectivity and convenience. This is where Edge AI comes in, turning this vision into reality. EdgeAI combines edge computing with AI [55], and is a paradigm that facilitates running ML algorithms on local edge devices such as sensors, Internet of Things (IoT) devices, or EdgeAI development platforms.

This project explores the integration of Edge AI technology with innovative space applications, focusing specifically on 'Autonomous Space Debris Monitoring and Classification in Low Earth Orbit.' The increase in space debris could be "storing up a catastrophic problem for humankind because the global economy is now increasingly reliant on space services for everyday activities [56]". As a result, effective

monitoring and classification of space debris is crucial to protect active satellites and operating infrastructure. "The larger objects tracked by ground-based global space surveillance networks are only a small proportion of an estimated 131 million pieces of hazardous space debris that cannot be monitored [56]". Relying on ground-based systems to monitor and classify space debris can result in delays, increased bandwidth usage, and will become unsustainable as the volume of space debris continues to grow. By developing an autonomous image classification system that operates directly on satellites, it is possible to achieve real-time processing, reduce the need to transmit a large amount of data, and make space missions more responsive.

This project aims to use cutting-edge hardware, such as the NVIDIA Jetson Nano, to develop a robust image classification system for space debris. The Jetson Nano will enable real-time processing of image classification tasks directly on the device (in an ideal scenario, the satellite), making it a complete piece of hardware for this purpose. By using a combination of real-world datasets and synthetic data generated through NVIDIA's Omniverse Replicator, the project aims to train ML models capable of detecting and classifying objects in space. The inclusion of synthetic data will ensure that the models are exposed to a wider range of scenarios, increasing their generalization capabilities. Initially, the EdgeImpulse platform will simplify the development process, enabling efficient training and deployment of these models on the Jetson Nano. With its no-code interface, EdgeImpulse streamlines data collection, model training, and deployment, removing much of the complexity associated with ML model setup and training.

D.3 Overall Aim

To develop and deploy an autonomous image classification system for space applications with NVIDIA's Jetson Nano Development Kit. The system will classify space debris around satellites by making use of real and synthetic data sets to train robust ML models.

D.4 Objectives

1. Literature Review:
   - Research existing image classification systems in space applications.
   - Research existing demonstrations of debris removal technology.
   - Identify specific requirements and constraints for autonomous on-board processing.
2. Dataset Collection and Synthetic Data Generation:
   - Gather existing data sets related to space debris.
   - Using NVIDIA's Omniverse Replicator to create synthetic datasets to enhance the diversity and volume of training data.
3. Model Development and Training:
   - Use EdgeImpulse to develop an initial Minimum Viable Product (MVP) model for image classification.

- Transition to manual model training using the JetPack SDK for more performance and customization.

4. Model Optimization and Validation:
   - Optimize the trained models for deployment on the chosen hardware platform.
   - Validate the performance of the model using both real and synthetic datasets to ensure accuracy and reliability.
5. Deployment and Testing:
   - Deploy the optimized models on the NVIDIA Jetson Nano.
   - Conduct testing to assess real-time classification capabilities and system stability.
6. Documentation and Presentation:
   - Document the development process, challenges, and solutions.
   - Prepare a comprehensive report and presentation showcasing the project outcomes.

E Initial Project Plan (Deliverable 1)

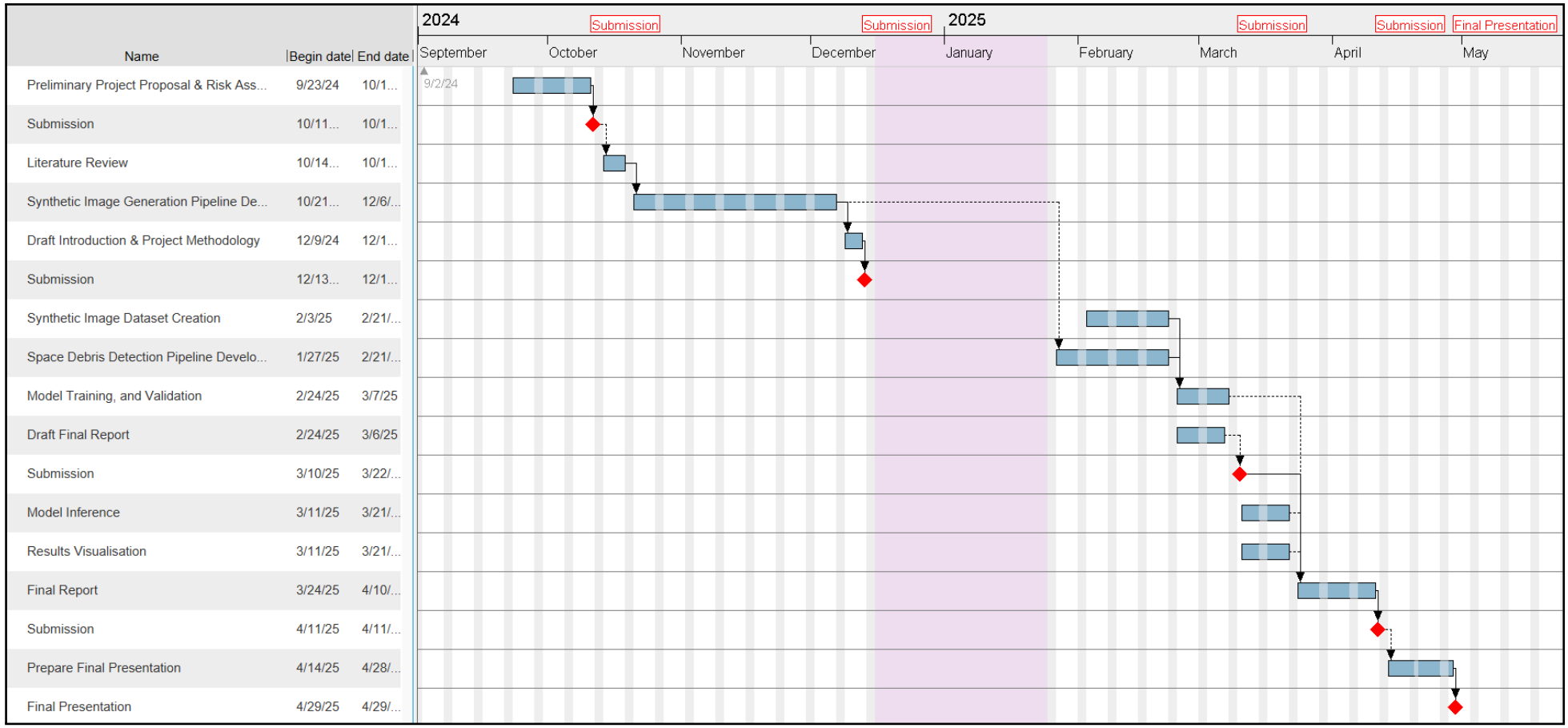


Fig. E.1. Gantt chart illustrating the detailed timeline and milestones.

F Risk Assessment

Table F.21. Project Risk Assessment

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of project timeline delays due to time management challenges during periods of high academic and career workload (e.g., exam preparation, coursework, and job applications).		✓				✓	6	Develop a detailed project schedule with built-in time buffers around known academic peaks. Prioritize tasks based on deadlines and begin critical tasks early to reduce last-minute pressure.	Despite a busy academic schedule and job application deadlines, the project timeline was largely maintained. Creating a buffer around peak academic weeks proved helpful. Minor delays occurred during exam preparation, but early starts on critical tasks ensured that final milestones were met on time. The mitigation was effective overall.
Risk of hardware components being unavailable in the online market.		✓			✓		4	Identify and secure backup suppliers early. Investigate equivalent hardware alternatives in case the primary components are unavailable, or in the worst-case scenario, pivot the project scope.	This risk did not occur. The required hardware component was available through one of the University's approved suppliers, eliminating the need to explore online marketplaces or alternatives. No delays or scope changes were necessary.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of hardware components not arriving on time for project implementation.		✓		✓			2	Order components early in the project timeline and identify alternative suppliers. Monitor shipment progress closely and prepare contingency plans using substitute hardware in case of delays.	The project required only one hardware component, which was available through one of the University's approved suppliers. As a result, the entire procurement process—including ordering and tracking—was handled by the University, and the component arrived within two days, much faster than anticipated. The risk did not materialize, and no delays occurred.
Risk of using non-compliant or improperly sourced data.			✓	✓			3	Use only open-access, synthetic, or properly licensed datasets. Verify dataset sources before use and consult University guidelines to ensure compliance with ethical and legal standards.	This risk did not materialize. All datasets used in the project were either synthetically generated or publicly available under open-access licenses. Dataset sources were verified during the initial planning phase to ensure compliance with ethical and copyright requirements.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of personal laptop being unable to handle required software load.			✓			✓	9	Run preliminary tests of all required software on the laptop early in the project. If performance issues arise, lease a more capable device from the University or use high-performance PCs in campus computer clusters.	This risk fully materialized early in the project. The laptop was unable to handle the required software load. The issue was escalated to the project supervisor, who helped arrange access to a more powerful PC. Although this caused a delay of approximately one week, the timeline was recovered, and the project progressed as planned thereafter. The mitigation was effective in resolving the issue without long-term disruption.
Risk of project data or code loss due to hardware failure or accidental deletion.			✓	✓			3	Regularly back up all work to cloud storage or external drives.	This risk did not materialize. All project files were stored on the University's provided cloud storage, with additional weekly backups maintained on a local hard drive.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of software incompatibility with personal laptop.			✓			✓	9	Ensure all required software is installed and tested on the laptop early in the project. Research alternative tools and prepare to use other machines if necessary.	This risk fully materialized during the initial setup phase. Some required software was incompatible with the personal laptop due to hardware limitations. The issue was resolved by switching to a more capable PC obtained through the University. Although it caused a brief delay, the mitigation was effective, and the project proceeded without further compatibility issues.
Risk of delays in obtaining necessary data sets or data sets being unavailable.	✓					✓	3	Research and source datasets early in the project. Consider using open-source datasets as alternatives.	This risk was anticipated and aligned with one of the project's core motivations—the lack of available datasets. While a complete dataset was not obtained, a small set of test images was acquired later in the project. These supported final model evaluation. Although the impact was limited, broader access to real datasets would have improved the validation process.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of inadequate project documentation, leading to a lack of evidence for methodology, results, and discussion.			✓	✓			3	Establish a structured documentation process that is updated regularly. Use version control for code and maintain a log of challenges, changes, and solutions throughout the project.	This risk did not materialize. A weekly log was maintained throughout the project, capturing methodologies, newly acquired knowledge, encountered challenges, and corresponding solutions. Version control was implemented using Git and GitHub, ensuring traceability and organization of code development.
Risk of a steeper-than-expected learning curve for software, potentially causing cascading delays.		✓				✓	6	Allocate sufficient time for learning new software early in the timeline. Use online tutorials, documentation, and community forums to support the learning process.	This risk materialized during the later stages of the project when the technical complexity increased. Progress slowed temporarily as a result. To overcome this, guidance was sought from professionals at STFC and Noctuai, peers from MMU, and several contacts on LinkedIn. These efforts helped clarify concepts and get the project back on track.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of project delays due to unanticipated technical challenges (e.g., debugging issues, software malfunctions).			✓			✓	9	Build flexibility into the timeline to account for unexpected technical issues. Maintain a habit of regular testing and troubleshooting as part of the workflow.	This risk materialized unexpectedly during the image rendering phase. The PC obtained from the University was unable to handle large-scale batch rendering. To address this, code-level workarounds were implemented to split the rendering process into smaller, manageable batches. Although the issue was unforeseen, it was resolved without causing major delays to the overall timeline.
Risk of failure to meet project deadlines for submissions.			✓	✓			3	Regularly review deadlines and milestones, updating progress on a weekly basis. Build in early submission windows to allow time for last-minute revisions.	This risk did not materialize. Work on each deliverable was started well in advance of the deadline, allowing most tasks to be completed early. The final days before submission were used for thorough proofreading and minor refinements, ensuring timely and high-quality submissions.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of eye strain, back strain, and repetitive strain injury from extended computer usage.			✓		✓		6	Follow ergonomic best practices, take regular breaks, and use blue light filters.	This risk materialized around six weeks into the project, resulting in noticeable neck discomfort. In response, a proper ergonomic setup was established by investing in a suitable chair and a desk of appropriate height. This improved comfort and reduced further strain during the remainder of the project.
Risk of sleep disruption and negative health impacts due to project workload combined with other academic commitments.			✓			✓	9	Apply effective time management strategies and prioritise sleep and well-being. Break large tasks into manageable segments to reduce the risk of overwork.	This risk did materialize during periods of peak workload when deadlines for multiple assignments coincided. While it was difficult to avoid entirely, efforts were made to manage tasks proactively and maintain a reasonable sleep schedule. Although some fatigue was experienced, the impact was contained, and overall health was not significantly compromised.

(Continued on next page)

(Continued from previous page)

Project Risk	**Severity**			**Likelihood**			**Score**	**Mitigation Measures**	**Outcome/Reflection**
	L	M	H	L	M	H			
Risk of stress or burnout due to overworking during critical project stages, especially near submission deadlines.			✓			✓	9	Build rest periods and relaxation time into the schedule. Recognise early signs of burnout and adjust workload or take breaks as needed.	This risk materialized during intense project phases, particularly close to submission deadlines. Although stress was unavoidable, conscious efforts were made to take short breaks and set aside time to unwind. These moments helped maintain focus and avoid the negative effects of burnout during those periods.
Risk of exceeding the allocated project budget of £180.		✓		✓			2	Track expenses throughout the project and prioritise essential purchases. Consult the project supervisor before making discretionary or high-cost purchases. Maintain a buffer in the budget for unforeseen requirements.	This risk did materialize, but in a controlled manner. The project budget of £180 was exceeded with supervisor approval for necessary additional expenses. The budget overrun was justified and did not negatively impact project progress or the outcomes of other students in the cohort.