POLITECNICO DI TORINO
Repository ISTITUZIONALE

Synthetic Dataset Generation for Car Detection: A Comparison Among Rendering Engines

*Original*
Synthetic Dataset Generation for Car Detection: A Comparison Among Rendering Engines / Manuri, F., Pindinello, L., Rossi, L.F., Sanna, A.. - ELETTRONICO. - (In corso di stampa). (19th International Conference on Computer Science and Information Technology Parigi (Francia) 25-27 Settembre 2026).

*Availability:*
This version is available at: 11583/3017711 since: 2026-11-09T16:24:26Z

*Publisher:*
ACM

*Published*
DOI:

*Terms of use:*

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

*Publisher copyright*
ACM postprint/Author's Accepted Manuscript

(Article begins on next page)

19 November 2026

# Synthetic Dataset Generation for Car Detection: A Comparison Among Rendering Engines

Federico Manuri*

Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino, federico.manuri@polito.it

Lorenzo Pindinello

Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino, lorenzo.pindinello@studenti.polito.it

Luca Francesco Rossi

Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino, lucafrancesco.rossi@polito.it

Andrea Sanna

Dipartimento di Automatica e Informatica, Politecnico di Torino, corso Duca degli Abruzzi 24, 10129 Torino, andrea.sanna@polito.it

The remarkable progress in deep learning for computer vision tasks over the past decade is inextricably linked to the availability of large, richly annotated datasets. The centrality of labeled data is particularly evident in supervised learning paradigms, where the quality of ground-truth annotations directly constrains the upper bound on achievable model performance, and the quantity and diversity of training samples determine a model's ability to generalize across deployment conditions. Synthetically generated imagery offers a compelling alternative to large-scale datasets from real-world imagery, directly addressing the scalability and annotation limitations of real-world data collection. Despite its practical advantages, synthetic data introduces a fundamental challenge: the domain gap between rendered images and real photographs. Although recent research on synthetic dataset generation, the impact of datasets created with different rendering engines on model accuracy for the car detection task, and the relationship between rendering times and model performance, has yet to be explored. This paper presents a synthetic dataset generation pipeline based on Blender and Unreal Engine, capable of producing datasets with varying levels of details by leveraging different rendering engines and automatically generating annotations for object detection and instance segmentation. The datasets have been used to train YOLO26 and evaluate how datasets generated with different rendering engines affect the model's performance. Unreal Engine's Path Tracer at 64 samples per pixel (spp) emerged as an efficient trade-off, offering comparable performance to high-quality settings at significantly reduced computational expense.

CCS CONCEPTS • Computing methodologies → Artificial intelligence • Computing methodologies → Computer graphics → Rendering • Computing methodologies → Artificial intelligence → Computer vision → Computer

vision problems → Object detection • Computing methodologies → Artificial intelligence → Computer vision → Computer vision problems → Image segmentation

**Additional Keywords and Phrases:** Synthetic Datasets, Rendering Engines, Car Detection.

## 1 INTRODUCTION

Computer Vision AI models enable the automatic analysis of visual data, playing a crucial role in various applications, from object detection to semantic segmentation and scene understanding. The remarkable progress achieved by recent deep learning approaches in computer vision tasks has been largely driven by the availability of large-scale annotated datasets. The empirical observation that model performance scales predictably with dataset size [24,28] has elevated data acquisition to a central challenge in the development of robust AI systems, often surpassing architectural innovation as the primary driver of accuracy gains. The centrality of labeled data is particularly evident in supervised learning paradigms, where the quality of ground-truth annotations directly constrains the upper bound on achievable model performance. However, creating high-quality labeled datasets remains a significant challenge. Manual annotation is labor-intensive and time-consuming, and may require domain expertise, especially in specialized applications.

Synthetic data has emerged as a promising solution to address these challenges. By leveraging simulation and rendering pipelines, synthetic data generation enables the creation of large-scale datasets with automatically generated annotations [23,18,9]. These pipelines provide full control over scene composition, object placement, lighting conditions, and camera parameters, allowing for systematic exploration of the data distribution. Moreover, synthetic environments enable the simulation of rare, dangerous, or otherwise inaccessible scenarios, which are critical for safety-sensitive applications. The level of details play a crucial role in reducing the domain gap between synthetic and real-world data distributions and improving model generalization. Recent work has shifted the focus from simply using synthetic data to explicitly measuring and minimizing the domain gap [26,14]. Rendering techniques based on physically-based rendering (PBR) principles simulate complex light transport phenomena, such as global illumination, reflection, and refraction, producing images that closely resemble real-world observations [20]. Consequently, there is a growing need for systematic investigations into how different levels of detail and controllability affect downstream learning performance. The proposed work addresses the following research questions:

- RQ1: How do datasets generated with different rendering engines affect the model performances for the car detection task?
- RQ2: Which is the best option to balance between dataset generation times and model performances?

In this work, we propose a synthetic dataset generation pipeline that leverages multiple rendering engines. Our approach enables the automated generation of “realistic”, precisely annotated image datasets, while maintaining fine-grained control over scene parameters. By providing empirical insights into these questions, we seek to advance the understanding of synthetic data generation and its role in modern computer vision pipelines.

## 2 RELATED WORKS

The use of synthetic data in autonomous driving research is well established, originating in efforts to replicate real-world benchmark distributions in controllable virtual environments. SYNTHIA [22] was among the first large-scale synthetic datasets specifically designed for urban scene understanding, providing pixel-level semantic annotations rendered from a game engine. However, the limited level of details of SYNTHIA's rendering pipeline introduced a substantial domain gap, limiting its practical utility as a training source for models intended for real-world deployment. The Virtual KITTI dataset [15] addressed this limitation by precisely replicating five tracking sequences from the real-world KITTI benchmark in a virtual environment, enabling a controlled comparison between synthetic and real-world performance. The VIPER dataset [21] instead addresses the problem of realistic scene composition to closely match the composition of corresponding physical environments, validating the realism of the collected data via perceptual experiments. However, this approach collects data from simulated worlds without access to their source code or content, thus limiting further enhancements or variations in both scene composition and rendering. The CARLA simulator [13] introduced an open-source urban driving environment that has since become one of the most widely adopted platforms for generating synthetic autonomous driving data, providing configurable traffic, weather, sensor models, and automatic annotation. Virtual KITTI 2 [11] substantially improved the visual fidelity of the original KITTI dataset by exploiting the Unity High Definition Render Pipeline (HDRP) with advanced post-processing techniques.

For semantic segmentation specifically, Synscapes [27] represented a significant advance in dataset quality, combining photorealistic street scene rendering with detailed per-pixel annotations and an explicit evaluation of how model performance varies with object distance, occlusion level, and viewing angle. A closely related contribution directly relevant to the present work is that of [19], who constructed a Blender-based pipeline for synthetic car detection from aerial and satellite imagery. Their approach manipulates car colors, sun position, and image resolution to match the distribution of real-world satellite imagery, generating a 5000-image dataset with bounding-box annotations. Boundless [25], a photo-realistic synthetic data generation system for enabling highly accurate object detection in dense urban streetscapes, has been compared with CARLA, demonstrating improvements in the model's object detection performance due to the images' photorealism. Arlovic et al. [10] instead explored synthetic dataset generation by comparing datasets generated with different 3D software in the industrial domain.

To the best of the author's knowledge, whereas multiple factors have been addressed in previous research works to understand how to create synthetic datasets comparable to realistic ones, a comparison among different rendering engines, especially in the domain of car object detection in urban environments, is novel. This paper addresses this problem by comparing rasterization-based rendering engines with ray tracing or path tracing approaches. All the considered rendering engines support physically-based materials.

## 3 METHODOLOGY

This section presents the proposed methodology for generating synthetic datasets for training and evaluating computer vision AI models. The process integrates diverse technologies and frameworks to ensure high-quality, controllable, and reproducible datasets. The technological components and tools utilized are described, followed by the overall design of the proposed system.

### 3.1 System Design

The development of a synthetic dataset generation pipeline required defining the 3D scene creation approach, the rendering engines, and the automatic labeling system. Blender [1], an open-source 3D creation suite, was selected for creating the 3D scene. Its key features include Python scripting, a modular architecture, and the Geometry Nodes feature, which is particularly valuable for procedural modeling and supports the automated generation of complex 3D layouts. Moreover, Blender's capabilities can be extended through a range of community- and commercial-based add-ons. Cycles [2], Blender's path tracing render engine, provides high-quality global illumination and denoising, whereas the EEVEE [3] render engine offers real-time rasterization-based rendering. Unreal Engine (UE) 5 [6] was adopted for its advanced real-time rendering capabilities: its deferred rendering pipeline, paired with the path tracing rendering capabilities, allows for flexible experimentation with different rendering approaches. BlenderProc2 [12], a Python-based framework built on Blender's API, was used to automate dataset generation and annotation. It provided modular functionality for loading assets, camera sampling, domain randomization, and ground truth annotation. This framework ensured scalability and reproducibility in generating synthetic datasets with controlled variations.

### 3.2 Proposed Pipeline

The proposed pipeline is depicted in Figure 1: the 3D scene is generated in Blender, whereas BlenderProc2 is used to define the lighting and camera positions. Once the scene is ready, BlenderProc2 automatically annotates each image. BlenderProc2 directly controls the rendering phase with Cycles, whereas a dedicated Blender script manages EEVEE rendering. Another custom script exports the scene for rendering with Unreal Engine.

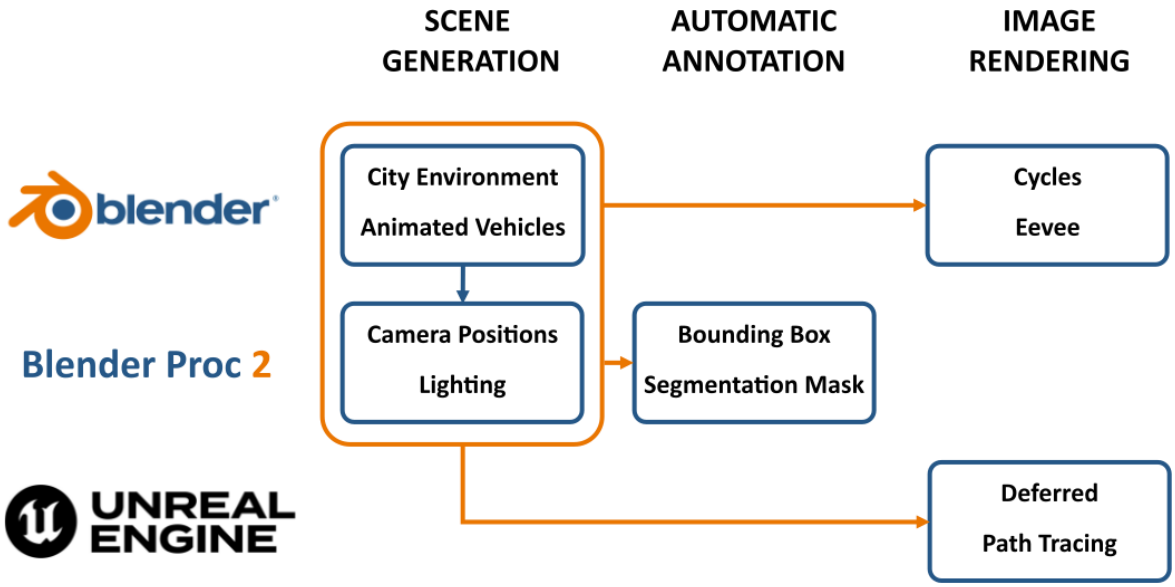



Figure 1 - The proposed pipeline for generating synthetic datasets with automatic annotation and different rendering engines.

### 3.3 Scene Generation

The first phase of the work involved creating the urban environment and defining scene dynamics within Blender. Since Blender serves as the entry point of the pipeline, the data structure defined here influences the quality of all subsequent processes, including export to Unreal Engine. To generate the urban environment, the The City Generator addon [4] was used, leveraging its procedural parameters to achieve a complete and functional city layout. The development focused on defining the scene's structural aspects and optimizing it.

#### *3.3.1 Urban Layout Definition*

Starting from a reference geometry, both streets and buildings were generated. A fundamental developmental choice was made to separate the control mesh for buildings from that of roads. This separation allowed independent intervention on buildings without interfering with road topology. For example, assigning different heights to buildings in the same block required loop cuts, which, when applied to road-based meshes, could be misinterpreted as new roads.

#### *3.3.2 Street Geometry and Vegetation Parametrization*

To achieve a dense, realistic urban scenario suitable for traffic simulation, the generation parameters were calibrated to maximize scene complexity. The street section was set to 22 meters wide, divided into four lanes, facilitating multi-level traffic scenarios in which front-line vehicles partially obscure those in opposing lanes, enriching the diversity of training data. Particular attention was paid to the generation of trees and green areas (parks), assigned procedurally based on specific face attributes in the base geometry. Using parameters such as tree density and random distribution, trees were instantiated along sidewalks and median strips.

#### *3.3.3 Optimization and Background Environment*

To balance geometric complexity with computational resources, the scene was structured into two levels of detail. The central area, where camera placement and traffic simulation occur, was modeled with high-polygonal density and rich details. To avoid an empty horizon during camera movements, streets and building were generated around the main area, composed of low-resolution geometry without traffic logic, serving solely as background fill to ensure coherent background information in dataset images.

#### *3.3.4 Vehicle Fleet Extension*

To ensure sufficient semantic variance in the dataset and prevent overfitting during neural network training, it was necessary to significantly expand the vehicle fleet beyond the default assets provided by The City Generator addon. This involved acquiring, optimizing, and integrating third-party 3D models (from repositories like CGTrader and Sketchfab) across various categories: standard cars (e.g., Alfa Romeo Giulietta, Fiat 127, Ford Focus), high-end vehicles (e.g., Audi A7, Mercedes-Benz Classe A), and commercial vehicles (e.g., ambulance, police car).

#### *3.3.5 Traffic Simulation*

Once the static environment and vehicle fleet were defined, traffic movement logic was configured using The City Generator's path following system. The road network was covered through procedural spline curve generation. A path amount parameter of 50 ensured uniform road coverage across the urban area. These

curves spanned all available lanes, including intersections and turns, ensuring vehicle presence in every portion of the virtual camera's field of view. To mitigate mesh penetration (where vehicles overlap), two critical parameters were calibrated: car distance, which defines the minimum safety distance between vehicle origins; and intersection distance, which regulates priority logic at intersections to prevent collisions on crossing paths. Moreover, to avoid robot-like or overly synchronized traffic behavior, a stochastic component was introduced in vehicle speed.

#### *3.3.6 Camera and Lighting*

Using the BlenderProc2 framework, the dataset generation pipeline was automated with a custom Python script that orchestrates Blender APIs. The goal was to transform the static, optimized scene into a dynamic, annotated dataset. Four distinct lighting conditions (sunny morning, overcast, partially cloudy, and sunset) were applied in batches to enhance dataset variance and improve model generalization. A stochastic camera placement algorithm was implemented to ensure valid viewpoints featuring traffic presence, with camera poses saved in both Blender and Unreal coordinate systems for cross-engine consistency. This approach, combined with the traffic simulation, enabled the automatic generation of the synthetic dataset starting from the 3D urban environment.

### 3.4 Automatic Annotation Generation

BlenderProc2's core lies in its ability to leverage rendering buffers to generate pixel-perfect annotations at zero cost, eliminating errors associated with manual annotation. By activating specific render passes, the system extracts auxiliary data perfectly aligned with the RGB image. When the urban scene was loaded, objects were classified into semantic categories, such as vehicles and background elements, with instances grouped by unique identifiers for coherent segmentation. Semantic Segmentation and Instance Segmentation are obtained by querying the unique object ID or material during rasterization or ray tracing, producing dense maps in which each pixel encodes the class or unique identifier of the visible object. Bounding boxes for object detection are computed beyond simple geometric projection by incorporating advanced visibility logic. The system filters out heavily occluded objects (<10% visibility) or those heavily truncated at image borders, which could introduce noise during training. Finally, additional data, such as depth maps and surface normals, are extracted, which are essential for multimodal geometric vision tasks. Figure 2 shows a sample image with the corresponding annotations.

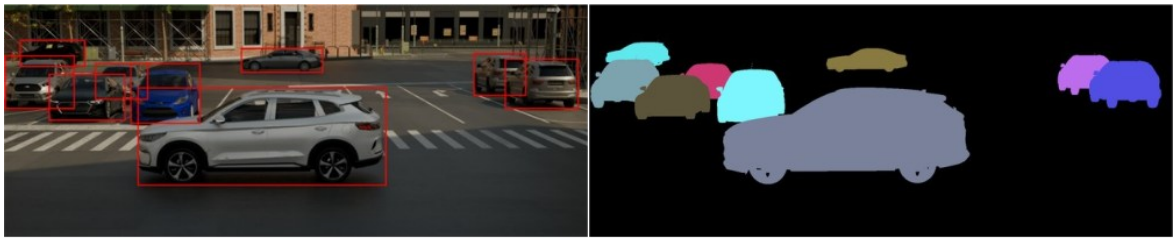


Figure 2 - A sample image rendered with Cycles with the corresponding object detection and instance segmentation annotations.

### 3.5 System Deployment

The computational demands of generating synthetic datasets using Path Tracing necessitated a high-performance workstation to handle the complexity of urban scenes in both Blender and Unreal Engine. The specifications included: an NVIDIA GeForce RTX 5090 (32GB VRAM) GPU; an Intel Core Ultra 9 285K CPU for robust processing capabilities; 64GB DDR5 RAM; and a 2TB SSD for efficient data storage during image batch saving. The rendering process relied on four rendering approaches: Blender Cycles, Blender EEVEE, Unreal Engine Deferred, and Unreal Engine Path Tracing. Figure 3 shows a sample image rendered with the four different engines.

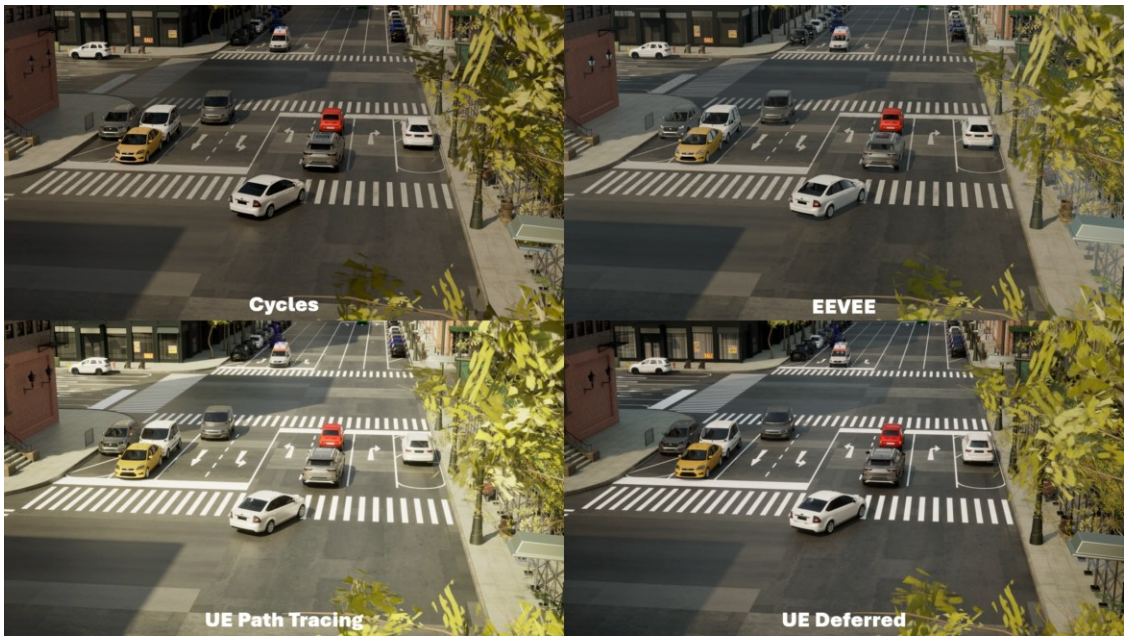



Figure 3- A sample image rendered with the four rendering approaches: Blender Cycles, Blender EEVEE, Unreal Engine Path Tracing, and Unreal Engine Deferred.

#### *3.5.1 Blender Cycles*

A dedicated Python script, leveraging the BlenderProc2 framework, performs the renderings, employing path tracing with 4096 samples per pixel as reference. Cycles, Blender's production-grade renderer, is based on the Path Tracing algorithm [16]. It simulates the physical behavior of light without compromising visual quality by solving the Light Transport Equation (LTE) using Monte Carlo methods [17]. For each pixel in the image plane, the engine shoots a series of primary rays into the scene. When a ray intersects a surface, the material response is calculated, and secondary rays (for reflection, refraction, or shadow) are generated in random directions. This process continues recursively until a light source is reached or a bounce limit is exceeded. The final pixel color is the weighted average of all sampled paths.

### *3.5.2 Blender EEVEE*

To evaluate computational efficiency between real-time rendering and Path Tracing, a parallel rendering process based on EEVEE was implemented using a custom Python script that leveraged Blender's API. The goal was to replicate the original dataset while maintaining identical geometric and dynamic consistency. EEVEE, Blender's real-time renderer, is based on Rasterization and prioritizes speed and interactivity, employing techniques similar to modern game engines. Unlike Ray Tracing, EEVEE does not simulate photon paths; it projects 3D geometry (triangles) directly onto the 2D screen plane. During this process, surface properties (color, depth, and normals) are written to specialized buffers (the G-Buffer). Subsequent passes combine these buffers to calculate final lighting. Limitations include simplified approaches to phenomena such as reflections and indirect lighting, which rely on Screen Space effects like SSR (Screen Space Reflections) and GTAO (Ground Truth Ambient Occlusion).

## 3.6 Export to Unreal Engine

Transferring the environment from Blender to Unreal Engine 5 presented one of the most critical technical challenges. Maintaining geometry, PBR materials, and procedural traffic animations required abandoning standard interchange formats in favor of more robust solutions and custom Python scripts. Initially, the FBX format was tested for asset exchange but proved unsuitable due to long import times and material misalignment. Thus, the glTF 2.0 format was adopted, ensuring near-perfect material correspondence and efficient management of object hierarchies. This procedure resulted in an Unreal Engine-ready asset library, with each vehicle correctly synchronized within the urban scene.

### *3.6.1 Unreal Engine Deferred*

Unreal Engine's default rendering mode is Deferred Shading, optimized for real-time performance. The "Deferred" approach separates geometry rasterization from lighting calculation, processing them in two passes. In the G-Buffer Pass, geometry is projected onto the screen, and material properties (Base Color, Normals, Roughness, Depth, Specular) are written into G-Buffers. In the Lighting Pass, the engine reads pixel data from G-Buffers and computes final lighting. This approach decouples geometric complexity from lighting computation, enabling the management of hundreds of dynamic lights. However, realism is limited by screen-space approximations (e.g., SSR and SSAO, Screen Space Ambient Occlusion), where reflections and ambient occlusion depend solely on what is visible in the current view, potentially introducing artifacts in the dataset.

### *3.6.2 Unreal Engine Path Tracing*

To overcome rasterization limitations, Unreal Engine incorporates an integrated Path Tracer accelerated by hardware (DXR) [5]. Like Blender Cycles, this mode uses Monte Carlo methods to solve the rendering equation. Instead of approximating light behavior, the engine shoots stochastic rays into the scene, accumulating samples over time to converge on physically correct results. Unlike Deferred Rendering, the Path Tracer natively handles Global Illumination and Reflections, computing multiple light bounces and realistic reflections, even for objects outside the view frustum, ensuring accurate physical shadows and refraction. Moreover, the image is generated by accumulating samples over time until noise is eliminated, making this mode suitable for producing visual ground truth comparable to that of traditional offline renderers.

### 3.7 Temporal Analysis

The computational efficiency of the rendering pipelines was rigorously analyzed on the specified workstation. Blender Cycles required approximately 146 hours to generate 10k images, with significant variance observed across different lighting conditions. In contrast, Blender EEVEE rendered the dataset in just 9.9 hours. Unreal Engine 5 Path Tracer with 4096 spp completed in 323.7 hours, whereas Path Tracer with 64 spp completed in only 5.14 hours, demonstrating superior efficiency. Finally, Unreal Engine Deferred finished in 6.22 hours, showcasing stable performance across varied lighting scenarios.

## 4 EVALUATION

### 4.1 Training on YOLO Models

The evaluation phase involved training two state-of-the-art models: YOLO26m (for object detection) and YOLO26m-seg (for instance segmentation) [7]. The primary objective was to determine how rendering fidelity and computational efficiency influence model performance. All experiments were conducted from scratch without pre-trained weights, ensuring that variations in performance metrics could be attributed solely to the characteristics of the synthetic data generated by different engines. Two types of datasets were created with each rendering engine: datasets with varying image counts (500, 1000, 5000, and 10000 images) were created to study the impact of dataset size on model performance, whereas time-based datasets were generated within fixed time budgets (1, 2, 5, 12, and 24 hours) to evaluate rendering efficiency. Further details on the tests with the different datasets, as well as YOLO hyperparameters, are available in the supplementary files (available at the following link https://tinyurl.com/r5krdscd), whereas results obtained with the 10k-image datasets are discussed here.

### 4.2 Metrics

The evaluation focused on three key metrics: precision, recall, and mean average precision (mAP). Training (80%) and validation (20%) splits were applied consistently across all datasets. The methodology employed two distinct phases: fixed-interval Sim-to-Real analysis and cross-engine validation. The first phase involved training the models exclusively on the synthetic datasets and evaluating their generalization capability by testing them on a real-world dataset: the Microsoft COCO dataset [8], car category, was used as a reference for standardizing performance measurements. This approach aimed to isolate the impact of rendering engines on model performance. The second phase focused on cross-engine validation using a dataset of 10000 images, with models trained on specific engines tested against datasets generated by other engines. This revealed insights into model generalization across synthetic domains.

### 4.3 Results

Table 1 presents the results obtained by training the AI model from scratch on the two AI models with 10000 images. In the object detection task, Cycles showed superior performance in balancing precision and recall, particularly for detecting vehicles in complex urban environments. However, UE PT 64 achieved results comparable to Cycles at a significantly lower computational cost, highlighting its efficiency for large-scale datasets. In the instance segmentation task, UE PT 4096 and UE Deferred achieved a comparable mAP50-95 score, whereas UE PT 4096 obtained the best recall score, underscoring the importance of high rendering

fidelity for capturing fine geometric details. On the other hand, EEVEE showed slightly conservative segmentation masks but struggled with complex reflections on vehicle surfaces.

Figures 4 and 5 present the results of cross-engine validation for the two AI models on 10000 images. Cross-engine validation revealed that models trained on data generated by Cycles and UE PT showed higher generalization accuracy across all engines. This suggested that Path Tracing, despite variations in software implementations, extracts consistent high-level features for semantic understanding.

Table 1 - Precision, recall, and mean average precision (mAP) for the object detection and instance segmentation tasks for the synthetic dataset evaluation.

| | Object Detection | | | | Instance Segmentation | | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | mAP50-95 | mAP50 | Precision | Recall | mAP50-95 | mAP50 | Precision | Recall |
| Cycles | 0.1205 | 0.2040 | 0.4387 | 0.2376 | 0.0987 | 0.1698 | 0.4090 | 0.1919 |
| Eevee | 0.0809 | 0.1442 | 0.3494 | 0.1762 | 0.0730 | 0.1361 | 0.3759 | 0.1708 |
| UE Deferred | 0.1103 | 0.1770 | 0.3705 | 0.1828 | 0.1090 | 0.1799 | 0.4609 | 0.1965 |
| UE PT 64 | 0.1037 | 0.1724 | 0.4560 | 0.1886 | 0.1035 | 0.1746 | 0.4183 | 0.2077 |
| UE PT 4096 | 0.1046 | 0.1753 | 0.4066 | 0.1953 | 0.1072 | 0.1835 | 0.438 | 0.2148 |

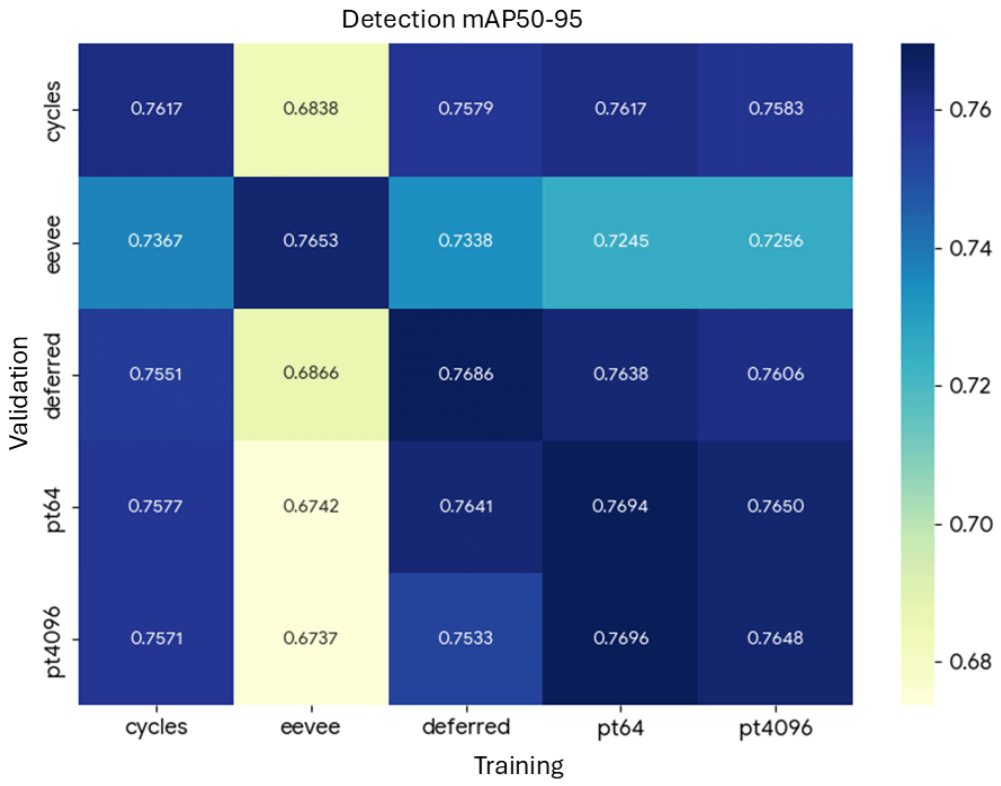



Figure 4 - Results of cross-engine validation for object detection.

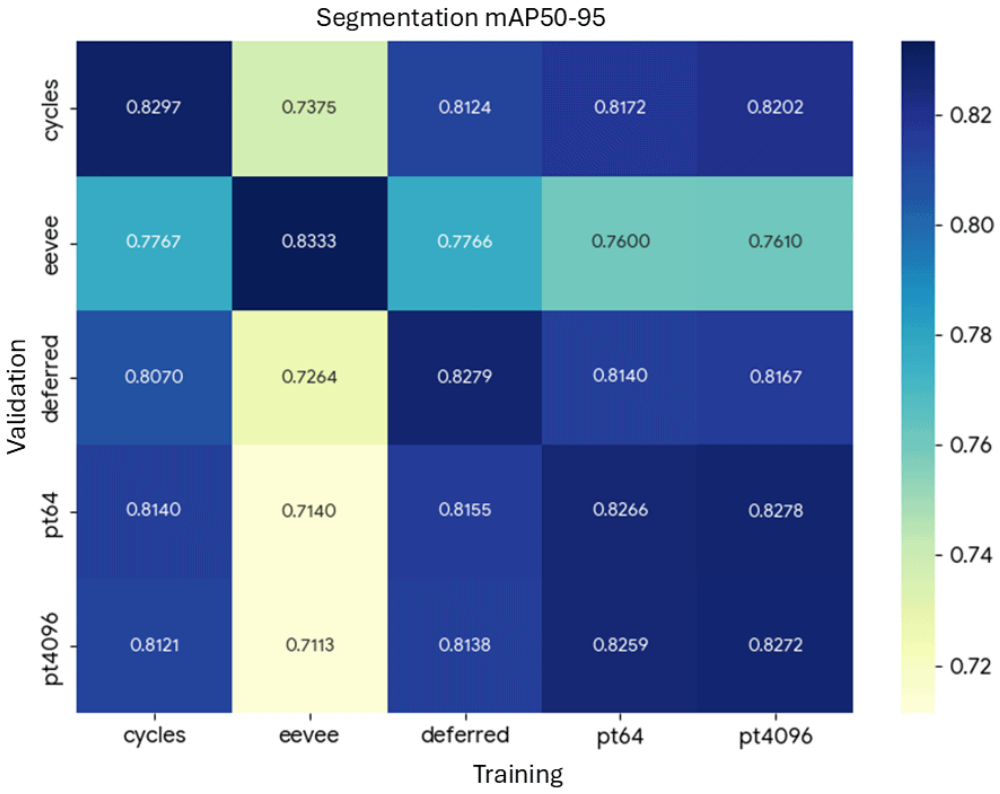



Figure 5 - Results of cross-engine validation for instance segmentation

### 4.4 Qualitative Comparison

To assess the effectiveness of the synthetic dataset generation pipeline, a direct qualitative comparison was conducted among images rendered using Blender Cycles (Path Tracing), Unreal Engine 5's Path Tracer (UE PT), EEVEE, and Unreal Engine's Deferred Renderer. The comparison aimed to evaluate how each engine handles "realistic" rendering, particularly in managing specular reflections on vehicle bodies and indirect light propagation. Cycles and UE PT demonstrated the highest fidelity in rendering, particularly for specular reflections and indirect lighting. Their ability to handle complex light interactions made them ideal for generating high-quality synthetic data. EEVEE and Unreal Deferred, while showing minor discrepancies in shadow contact and glass refraction, maintained semantic consistency in object placement and vehicle positioning, validating the effectiveness of the synchronization scripts.

### 4.5 Discussion and Limitations

The comparative analysis highlights a balance between visual fidelity and productive scalability within the experimental framework adopted. While path tracing engines with high sampling rates produce qualitatively superior data, the computational cost involved limits the volume of data that can be generated within the same temporal window. Overall, as shown in Table 1, datasets generated from the same scene but rendered with different rendering engines affect the precision, recall, and mean average precision (mAP) for car detection and instance segmentation (RQ1). However, since these differences are quite limited, the temporal analysis results suggest opting for the rendering engine that offers the highest performance while maintaining comparable time creation. In this context, the UE Path Tracer 64 dataset provided an effective trade-off between physical accuracy and generation speed (RQ2). However, it is important to note that these findings emerge from a restricted setting and cannot be uncritically extended to different application domains. Although this approach yielded the best results in our experiments, these results are not statistically significant per se. Table 2 presents the results obtained by performing an additional 300-epoch fine-tuning on 1000 real images from the COCO dataset, after pre-training the two AI models on 10000 synthetic images from the UE PT 64 dataset. The proposed baseline consists of the two AI models trained solely on the COCO 10k dataset. Results show that fine-tuning with a limited dataset of real-world images greatly improves the model's performance, yielding results almost as good as those of models trained exclusively on large real-world datasets. Moreover, these results shows that the dataset's statistical diversity is more critical than the visual fidelity. The datasets created for this study and the scripts developed to implement the proposed pipeline will be made available on request.

Table 2 : Precision, recall, and mean average precision (mAP) for the object detection and instance segmentation tasks for the fine-tuning evaluation. UE PT 64 ft consists of 10k synthetic images + finetuning over 1k real images randomly extracted from the COCO dataset.

| | Object Detection | | | | Instance Segmentation | | | |
|---|---|---|---|---|---|---|---|---|
| Dataset | mAP50-95 | mAP50 | Precision | Recall | mAP50-95 | mAP50 | Precision | Recall |
| UE PT 64 ft | 0.3252 | 0.5168 | 0.7000 | 0.4624 | 0.3340 | 0.5466 | 0.7103 | 0.4868 |
| COCO | 0.4282 | 0.6500 | 0.7639 | 0.5962 | 0.4361 | 0.6611 | 0.7531 | 0.6057 |

## 5 CONCLUSIONS AND FUTURE WORKS

This paper presents a synthetic dataset generation pipeline based on Blender and Unreal Engine, capable of producing datasets with varying levels of details by leveraging different rendering engines and automatically generating annotations for object detection and instance segmentation. The evaluation demonstrated that while high-fidelity rendering engines like Cycles and UE PT provide superior photorealistic output, their computational cost limits practical applications for large-scale dataset generation. Unreal Engine's Path Tracer with moderate sampling (64 spp) emerged as an efficient compromise, offering comparable performance to higher-fidelity settings at significantly reduced computational expense. These findings underscore the importance of balancing rendering quality and efficiency when generating synthetic data for AI training in the automotive domain.

Future work should focus on enhancing the dataset's statistical diversity, which has proven more critical than the visual fidelity of individual pixels. This includes creating large, diverse urban settings to replicate a wide array of critical scenarios and extending environmental modeling to incorporate highway, rural, and natural landscapes. Scene composition should be further enhanced by including pedestrians and cyclists, and by expanding the vehicle fleet. Adding challenging weather conditions, such as rain, would introduce complexities related to lens refraction and wet asphalt reflectivity, whereas night conditions would test the model's ability to manage point light sources and strong contrasts. Lastly, implementing systematic post-processing effects such as motion blur, depth-of-field blur, and sensor noise, which can be obtained with Blender compositing capabilities, would make synthetic data less "perfect" and more akin to noisy real-world acquisitions, thereby improving model robustness under difficult inference conditions.

## REFERENCES


[1] Blender. https://www.blender.org/

[2] Blender Cycles. https://www.cycles-renderer.org/

[3] EEVEE. https://docs.blender.org/manual/en/latest/render/eevee/introduction.html

[4] The City Generator. https://blender-addons.org/the-city-generator/

[5] UE Path Tracer. https://dev.epicgames.com/documentation/unreal-engine/path-tracer-in-unreal-engine

[6] Unreal Engine homepage. https://www.unrealengine.com/

[7] YOlO 26 homepage. https://docs.ultralytics.com/models/yolo26

[8] Coco dataset webpage. https://cocodataset.org/ (2021), accessed 2026-04-30

[9] Synthetic data for computer vision (SynData4CV) workshop at CVPR 2024. https://syndata4cv.github.io/cvpr2024.html (2024), accessed 2026-04-30

[10] Arlovic, M., Damjanovic, D., Hrzic, F., Balen, J.: Synthetic dataset generation methods for computer vision application. In: 2024 International Conference on Smart Systems and Technologies (SST). pp. 69–74. IEEE (2024)

[11] Cabon, Y., Murray, N., Humenberger, M.: Virtual kitti 2. arXiv preprint arXiv:2001.10773 (2020)

[12] Denninger, M., Winkelbauer, D., Sundermeyer, M., Boerdijk, W., Knauer, M., Strobl, K.H., Humt, M., Triebel, R.: Blenderproc2: A procedural pipeline for photorealistic rendering. Journal of Open Source Software 8(82), 4901 (2023). https://doi.org/10.21105/joss.04901

[13] Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: Carla: An open urban driving simulator. In: Conference on robot learning. pp. 1–16. PMLR (2017)

[14] Duc, N., Lai, Y.L., Madlindl, P., Zhu, X., Schwab, B., Wysocki, O., Hoegner, L., Kolbe, T.H.: Mind the domain gap: Measuring the domain gap between real-world and synthetic point clouds for automated driving development. arXiv preprint arXiv:2505.17959 (2025)

[15] Gaidon, A., Wang, Q., Cabon, Y., Vig, E.: Virtual worlds as proxy for multi-object tracking analysis. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 4340–4349 (2016)

[16] Jensen, H.W.: Global illumination using photon maps. In: Eurographics workshop on Rendering techniques. pp. 21–30. Springer (1996)

[17] Kalos, M.H., Whitlock, P.A.: Monte carlo methods. John Wiley & Sons (2008)

[18] Nikolenko, S.I., et al.: Synthetic data for deep learning, vol. 174. Springer (2021)

[19] Orić, M., Galić, V., Novoselnik, F.: Synthetic car dataset for vehicle detection: Integrating aerial and satellite imagery. Data in brief 53, 110105 (2024)

[20] Rendering, W.P.B.: Physically-based rendering. Procedia IUTAM 13(127-137), 3 (2015)

[21] Richter, S.R., Hayder, Z., Koltun, V.: Playing for benchmarks. In: Proceedings of the IEEE international conference on computer vision. pp. 2213–2222 (2017)

[22] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., Lopez, A.M.: The Synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3234–3243 (2016)

[23] Shorten, C., Khoshgoftaar, T.M.: A survey on image data augmentation for deep learning. Journal of big data 6(1), 1–48 (2019)

[24] Sun, C., Shrivastava, A., Singh, S., Gupta, A.: Revisiting unreasonable effectiveness of data in deep learning era. In: Proceedings of the IEEE international conference on computer vision. pp. 843–852 (2017)

[25] Turkcan, M.K., Li, Y., Zang, C., Ghaderi, J., Zussman, G., Kostic, Z.: Boundless: Generating photorealistic synthetic data for object detection in urban streetscapes. arXiv preprint arXiv:2409.03022 (2024)

[26] Werda, M.S., Taibi, H., Kouiss, K., Chebak, A., Halima, S.B., Decottignies, M., Dilliott, C.: Towards minimizing domain gap when using synthetic data in automotive vision control applications. IFAC-PapersOnLine 58(19), 522–527 (2024)

[27] Wrenninge, M., Unger, J.: Synscapes: A photorealistic synthetic dataset for street scene parsing. arXiv preprint arXiv:1810.08705 (2018)

[28] Zhai, X., Kolesnikov, A., Houlsby, N., Beyer, L.: Scaling vision transformers. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 12104–12113 (2022)

# Authors' Background

| **Name** | Email | Position (Prof , Assoc. Prof. etc.) | **Research Field** | Personal Website |
|---|---|---|---|---|
| Federico Manuri | **federico.manuri@polito .it** | **Assistant Professor** | **Computer Vision, Extended Reality, Human-Machine Interaction** | **https://www.polito.it/en/staff?p=federico.manuri** |
| Lorenzo Pindinello | **lorenzo.pindinello@studenti.polito.it** | **Former Master Degree Student** | **Computer Science, Artificial Intelligence** | |
| Luca Francesco Rossi | **lucafrancesco.rossi@polito.it** | **PhD Student** | **Computer Vision, Artificial Intelligence** | **https://www.polito.it/en/staff?p=lucafrancesco.rossi** |
| Andrea Sanna | **andrea.sanna@polito.it** | **Full Professor** | **Computer Vision, Extended Reality, Human-Machine Interaction** | **https://www.polito.it/en/staff?p=andrea.sanna** |