POLITECNICO DI TORINO

Master's Degree in Computer Engineering

Master's Degree Thesis

Enhance Security with Robots and Artificial Intelligence

Supervisors

Prof. Giuseppe Bruno AVERTA

PhD. Francesca PISTILLI

Ing. Claudio CHIEPPA

Candidate

Francesco P. CARMONE

April 2025

Alla mia famiglia

Summary

Ensuring security in critical infrastructures, industrial sites, and remote areas requires advanced AI-driven solutions, as traditional methods—such as fixed cameras and human patrols—suffer from limitations in adaptability, coverage, and reliability. A key component of any AI-based surveillance system is the dataset used for training, as the accuracy and robustness of anomaly detection models heavily depend on the quality and diversity of training data. This thesis focuses on the development of an AI-powered perception system capable of real-time object detection and anomaly recognition, addressing the critical challenge of dataset acquisition. To overcome the scarcity of labeled data, a synthetic dataset was generated using the AI-based model `FLUX.1-schnell`. This approach ensured a diverse and representative dataset, improving the generalization capabilities of the neural network in complex surveillance scenarios. The perception system runs on an NVIDIA Jetson Orin Nano 8GB and employs NVIDIA DeepStream to efficiently process video streams in real time. Object detection is performed using a YOLO-based neural network, with optimized inference running on the edge device. The processed data, including detected objects, bounding boxes, and timestamps, is transmitted via the Kafka communication protocol to a central server, where it is visualized through a web application. This enables real-time monitoring and historical review of detected events, improving situational awareness and response times.

By focusing on AI-driven perception and dataset generation, this work contributes to the advancement of autonomous surveillance systems. The integration of synthetic data augmentation, edge AI processing, and real-time streaming enhances the reliability and scalability of automated security solutions, laying the groundwork for future developments in intelligent monitoring technologies.

Acknowledgements

La stesura di questo documento non segna solo la conclusione di sei mesi di tesi in azienda, ma la chiusura di un capitolo molto più ampio, iniziato nel tardo 2018 e giunto oggi al suo epilogo, nell'aprile 2025. È stato un percorso segnato da difficoltà immense, rimorsi, incertezze e angosce. Un cammino fatto di ferrea disciplina, routine, bilanci e strategia.

Desidero innanzitutto ringraziare il prof. *Giuseppe Bruno Averta* per il suo immediato sostegno. Un sentito ringraziamento va a *Claudio Chieppa* e a Concept Reply per la fiducia accordatami nell'utilizzo di tecnologie avanzate e strumenti di valore, permettendomi di esplorare soluzioni concrete con responsabilità e libertà.

Rivolgo un pensiero speciale a mia *madre*, mio *padre* e mia sorella *Clara* per essere sempre rimasti al mio fianco, anche nei momenti più bui, con un sostegno incondizionato, tanto materiale quanto emotivo, e per cui provo un profondo ma silenzioso affetto.

A Isabel, il cui affetto genuino mi ha restituito una stabilità a lungo cercata. A ICARUS e ai suoi membri, i primi a mostrarmi cosa significhi affrontare problemi reali, lontani dai tomi e dai discorsi altisonanti, dove il valore di una soluzione si misura nella sua capacità di far funzionare un sistema. A *Claudio Macaluso*, che negli ultimi mesi mi ha fatto provare cosa significhi avere un fratello. A *Samuele Giannetto* e *Simone Voto*, sempre presenti nel mio periodo in azienda, pronti ad ascoltare i miei deliri e voli pindarici. A *Simone Carena* e *Ludovica Mazzucco*, con cui ho condiviso giornate estenuanti dedicate ai progetti. Ai Re Magi – *Francesca Fusco, Michele Ferrero, Alessio Cappello, Angelo Gennuso, Marco Rosa Gobbo (Marcorogo), Edoardo Cavallotti, Federico Ghiglione, Letizia Licitra* – non solo per la loro amicizia, ma per avermi accolto con naturalezza in un momento in cui ne avevo più bisogno. Dopo un anno a Pisa in cui tutto ciò che davo per certo – accademicamente e socialmente – era crollato, con loro ho ritrovato la leggerezza di costruire un gruppo senza sforzo, senza pretese.

E infine, a te, lettore, che hai dedicato il tuo tempo a sfogliare queste pagine.

Table of Contents

Chapter 1

Introduction

1.1 Goal and Problem Statement

Security has always been a fundamental human concern, leading to the development of various protective measures throughout history. Today, ensuring the safety of critical infrastructures, industrial plants, and remote areas remains a pressing challenge. Traditional patrolling methods, relying on human guards and fixed surveillance cameras, suffer from limitations in coverage, reaction time, and analytical capacity. Automating repetitive security tasks through AI-driven surveillance is a natural evolution, offering continuous 24/7 monitoring, minimizing human errors, and improving threat detection. However, the effectiveness of such AI systems hinges on one crucial factor: the availability of high-quality training data.

This thesis focuses on developing an autonomous patrolling system that leverages AI-powered perception to detect predefined objects and behaviors in real time. The core challenge lies in training an AI model capable of accurately identifying anomalies, unauthorized individuals, or security threats. Achieving this requires a well-curated dataset that captures the complexity of real-world surveillance scenarios. However, collecting and labeling large-scale security datasets is costly, time-consuming, and often impractical due to privacy concerns and the rarity of real security incidents. To overcome this, a synthetic dataset was generated using AI-based techniques, ensuring a diverse and comprehensive training set without the limitations of traditional data collection. The proposed system integrates an AI inference pipeline running on an NVIDIA Jetson Orin Nano 8GB, leveraging NVIDIA DeepStream for real-time video analysis. Object detection is performed using a YOLO-based neural network, trained on the synthetically generated dataset to improve generalization across different environments. Once detected, security events—such as anomalies or intrusions—are transmitted via the Kafka communication protocol to a remote monitoring system. A web-based interface enables

security personnel to visualize real-time alerts, access historical data, and respond effectively to potential threats. By focusing on AI-driven perception and synthetic data generation, this thesis addresses the fundamental challenge of training reliable security models without the need for extensive real-world data collection. The resulting system provides a scalable, adaptable, and cost-efficient surveillance framework that enhances security operations while reducing reliance on manual patrols.

1.2 State of the Art

Security robots have been increasingly adopted in recent years to enhance surveillance capabilities in urban environments. In 2017, the K5 Model robot was deployed in Los Angeles, CA, contributing to a reduction in crime incident reports and an increase in arrests. This demonstrates the potential effectiveness of autonomous security systems in real-world scenarios [1]. A more recent study [2] presents a security robot with a highly similar architecture to the one described in this thesis, integrating a ROS2 layer for autonomous navigation. The system leverages computer vision techniques to detect violent behavior, recognize weapons and hazardous objects, and identify individuals. Upon detecting a critical incident, the robot triggers an alarm and transmits the relevant data to a central security server, where a web-based interface enables security personnel to monitor events in real time. The detection pipeline is built upon the `YOLOv8` neural network, a widely adopted architecture for real-time object detection. Another crucial aspect of these autonomous security systems is their ability to log and communicate incident reports. While functional, such an approach raises concerns regarding scalability and maintainability, as it tightly couples the detection module with the communication layer. A more flexible and modular solution may be preferable for long-term deployment and integration with existing security infrastructures. One of the key challenges in designing a security system is defining what constitutes an *intruder* or a security threat. Previous works, such as [2] and [3], have explored this issue, utilizing various approaches that range from vision-based detection to audio analysis. However, defining an intruder remains a non-trivial problem, highly dependent on environmental context and operational constraints. A fundamental contribution of this thesis lies in the dataset generation process, which serves as the backbone for training and validating the detection models. A novel workflow was developed using ComfyUI to automate the generation of large-scale synthetic datasets tailored to specific security tasks. This workflow has been tested extensively in the context of detecting broken fences, a scenario relevant to perimeter security. However, due to its modularity, the same pipeline can be easily adapted to different use cases with minimal effort. Since the entire data generation and

labeling process has been systematized, switching to a new task would require less than a week, making this approach highly scalable and flexible for various applications.

1.3 Content Overview

This thesis explores the implementation of an AI-based detection system, integrating computer vision techniques with an efficient processing pipeline. The System Components Overview (2) chapter presents the hardware and software used, detailing the rationale behind each choice and discussing potential alternatives for specific components. The Computer Vision (3) chapter provides a foundational understanding of neural networks and their role in object detection. It highlights the importance of high-quality datasets, the risks of overfitting, and strategies for dataset preparation. Additionally, it explores how generative AI can be leveraged to augment training data. The DeepStream Pipeline (4) chapter details the inference pipeline, responsible for processing input sources, running AI-based detection, and transmitting results. It examines the different configurations and optimizations applied to improve efficiency. The Results (5) chapter evaluates the system's performance, analyzing the neural network's effectiveness across different setups involving one, two, and three cameras. It also assesses resource consumption, including energy, memory, and video memory, ensuring compliance with system constraints. The Conclusions and Future Work chapter (5) summarizes the key findings and discusses potential improvements, outlining directions for future research and development.

Chapter 2

System Components Overview

This chapter describes the technologies used throughout the thesis, which are considered hardware and software, as well as why certain choices were made and what component can be replaced.

2.1 Hardware

The key elements main of this system are: a camera to capture a video stream, a board to process such data and has internet access for external communication, and a mobile robotic platform. Each component is widely available from numerous manufacturers, offering similar quality at competitive prices. The system is platform-agnostic, meaning it can be integrated with any mobile robot as long as it is powered and has internet access. A Jetson device was chosen specifically in order to leverage the DeepStream SDK (see 2.2) and its advantages for this application.

2.1.1 Intel RealSense d435i

An ideal video camera would support multiple resolutions, have different framerates to allow for experiments in various setups, have a low energy consumption, high ISO sensitivity to operate in low-visibility conditions, while maintaining a competitive price.

Table 2.1: Intel RealSense d435i Colour Spaces and Resolutions

Color Space	640x480	1280x720	1920x1080
RGB	✓	✓	✓
YUV2	✓	✓	
Infrared (IR)	✓	✓	✓
Depth	✓	✓	✓

Throughout the thesis, the pipeline developed utilzes the `YUV2` colour space, ignoring the other webcam capabilities.

Table 2.2: Intel RealSense d435i Supported Resolutions and FPS for the YUV2 colour space

FPS	424x240	640x480	848x480	960x540	1280x720	1920x1080
60	✓	✓	✓	✓	✓	
30	✓	✓	✓	✓	✓	✓
15	✓	✓	✓	✓	✓	✓
6	✓	✓	✓	✓	✓	✓

2.1.2 Jackal UGV Rover

Jackal 2.1 is a small, fast, entry-level field robotics research platform. It has an onboard computer, GPS and IMU fully integrated with ROS for out-of-the-box autonomous capability. As with all Clearpath robots, Jackal is plug-and-play compatible with a huge list of robot accessories to quickly expand your research and development.

Figure 2.1: Clearpath Jackal at Reply Area 42

Size and Weight	
External Dimensions	508 x 430 x 250 mm
Internal Dimensions	250 x 100 x 85 mm
Weight	17 kg
Maximum Payload	20 kg

Table 2.3: Clearpath's Jackal Physical specifications

Speed and Performance	
Max Speed	2.0 m/s
Run Time (Basic Usage)	4 hours User Power
5V at 5A, 12V at 10A, 24V at 20A	
Drivers and API	ROS 1 Noetic, ROS 2 Humble

Table 2.4: Clearpath Jackal Performance specifications

2.1.3 UniTree Go2

Go2 is an entry level robotic platform with canine shape and dynamics, it's equipped with a Standard Ultra-wide 4D LIDAR for maps generation, a battery capacity increased to 8,000mAh [4].

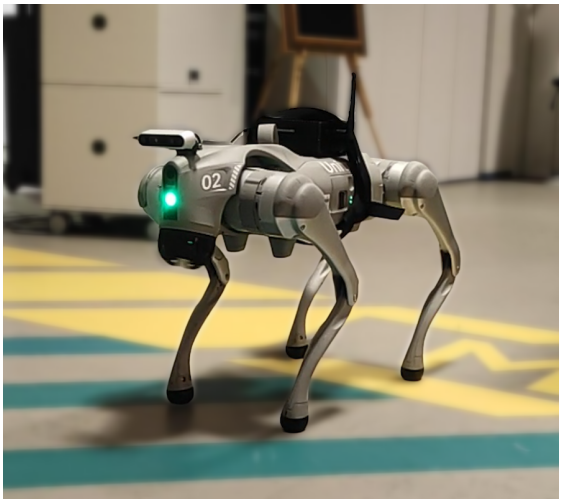


Figure 2.2: Unitree Go2 at Reply Area 42

Size and Weight	
External Dimensions	70×31×400mm
Weight	15 kg
Maximum Payload	10 kg

Table 2.5: Unitree Go2 Physical specifications [5]

Speed and Performance	
Max Speed	3.7 m/s
Run Time (Basic Usage)[1]	1.5 hours
Maximum Slope	30°
Drivers and API	Proprietary[2]

Table 2.6: Unitree performance specifications of the system [5]

During the measurement process, the Jackal encountered a technical issue that could not be resolved within the timeframe of this thesis. Consequently, development and testing had to be continued on an alternative platform. Due to the nature of this module, the migration was seamless and required no additional code modifications.

2.1.4 Workstation

Training, fine-tuning, or performing inference with a neural network is a computationally demanding task that requires powerful GPUs, large-capacity RAM, and multi-threaded CPUs to handle the complex matrix operations optimized by the graphics card. Laptops are generally unsuitable for such workloads. Therefore, I relied on a high-performance workstation for these tasks. The workstation used was equipped with 64GB of DDR5 RAM, an NVIDIA RTX 4090 GPU, and a 13th-generation Intel Core i7 processor. This system was utilized both for training the neural network (5.1) and for running Flux.1 to generate images (3.3.1).

2.1.5 Laptop

Although the Jetson runs a fully-fledged Ubuntu Desktop, it is neither intended nor well-suited for desktop use. Instead, the majority of the code was developed on a ThinkBook (2023) equipped with an AMD Ryzen 7 7730U processor with Radeon Graphics and 32GB of RAM, running an Arch Linux distribution. This same machine was also used to evaluate the performance of the AI model with the .pt format.

[2]Available ROS2 Integration. Nobody in the company managed to get them to run though.

2.2 The NVIDIA DeepStream SDK

The NVIDIA DeepStream SDK is a comprehensive streaming analytics toolkit designed for AI-driven multisensor processing, video, audio, and image analysis. Built on GStreamer, an open-source framework widely adopted for its compatibility and flexibility in processing various multimedia streams, DeepStream provides powerful capabilities for real-time analytics. To use DeepStream in an application, developers construct a pipeline, which can be implemented through Python or C++ code or configured via key-file format configuration files based on the freedesktop specifications. These files allow enabling or disabling components, modifying their properties, and customizing other settings, as documented in the official guides. DeepStream excels at handling multiple video streams simultaneously and achieves high inference speeds. Rather than processing a single frame at a time, it batches multiple frames or bjects together, optimizing processing efficiency. This batching leverages the parallel computing power of GPUs, reducing latency and increasing throughput. However, for applications that do not required continuous data acquisition or rely on batch processing of static datasets—such as pre-recorded videos or images—DeepStream might introduce unnecessary complexity and overhead. In such cases, a simpler alternative like OpenCV may be more suitable.

2.2.1 DeepStream vs GStreamer + OpenCV

GStreamer is an open-source multimedia framework widely used in the GNU software stack and the open-source ecosystem. DeepStream builds upon GStreamer, extending its capabilities with specialized plugins optimized for AI inference. Unlike DeepStream, GStreamer alone lacks native support for inference engines and must be combined with OpenCV to process video frames through AI models. OpenCV is a versatile open-source library that provides a broad set of tools for image and video processing. A key advantage of OpenCV is its flexibility in handling custom data loaders, enabling tailored video input and preprocessing pipelines. Its open-source nature allows seamless integration with other libraries, full access to source code, and easier debugging and customization. However, for real-time inference on NVIDIA hardware, DeepStream is significantly more efficient. DeepStream is specifically optimized for GPU acceleration, whereas OpenCV, though highly flexible, does not offer the same level of performance in high-throughput AI tasks.

2.2.2 DeepStream vs Pipeless

Pipeless is an open-source framework designed to simplify the development of computer vision applications by abstracting the complexity of GStreamer. Unlike DeepStream, which requires developers to manually construct GStreamer pipelines,

Pipeless allows the implementation of Python hooks that are automatically triggered at the right time, significantly reducing the development effort. With its Apache 2.0 license, Pipeless offers full access to the source code for debugging and customization. It features an extensible plugin system and supports distributed frame processing, enabling scalability without GPUs by adding workers on different machines. While DeepStream provides optimized performance for real-time, GPU-accelerated applications, Pipeless focuses on developer accessibility, allowing functional applications to be created in minutes with minimal code. DeepStream has a steeper learning curve and closed-source limitations, whereas Pipeless is open-source and highly extensible. Scalability in Pipeless is built-in, whereas DeepStream requires additional configurations. For rapid prototyping or non-GPU-dependent projects, Pipeless offers a more accessible alternative.

2.2.3 Conclusions on DeepStream

A simple test setup was designed to compare DeepStream and Pipeless (GStreamer + OpenCV) using an identical hardware configuration and the same `YoloV11m` model, with weights provided by Ultralytics. The input video consisted of a 12-second clip, where each object class appeared sequentially, ensuring a consistent evaluation across both frameworks (Figure 4.1).

Metric	DeepStream	Pipeless (GStreamer + OpenCV)
Average FPS	29.99	29.99
CPU Usage (%)	16	97
GPU Usage (%)	91	15[3]

Table 2.7: Comparison between DeepStream and Pipeless (GStreamer + OpenCV) using a basic pipeline. Although inference time and speed were not directly measured, GPU usage indicates that DeepStream offloads most computations to the GPU, whereas Pipeless relies heavily on CPU processing.

DeepStream's superior speed stems from its GPU acceleration and optimized parallel processing, making it highly suitable for real-time applications. However, its main drawback is its exclusive reliance on NVIDIA hardware, rendering it unusable on non-NVIDIA platforms. Given the available hardware and the real-time requirements of this thesis, DeepStream was selected for its efficiency in handling inference workloads and delivering low-latency object detection.

[3]Similar results were observed when measuring idle power consumption with the Jetson powered on.

2.3 Jetson Orin Nano and Nvidia Jetpack

NVIDIA Jetson is a series of embedded computing boards from Nvidia. The Jetson TK1, TX1 and TX2 models all carry a Tegra processor (or SoC) from Nvidia that integrates an ARM architecture central processing unit (CPU). Jetson is a low-power system and is designed for accelerating machine learning applications [6]. Given the mentioned features and qualities, combined with the accessible price, the choice fell on the NVIDIA Jetson Orin Nano 8GB. The alternatives were unnecessarily more expansive for qualities that weren't require.

AI Performance	67 TOPS
GPU	1024-core NVIDIA Ampere arch, GPU, 32 Tensor Cores
GPU Max Frequency	1020 MHz

Table 2.8: Technical Specifications for Jetson Orin Nano 8GB

The physical hardware comes along with the DeepStream SDK [7] and JetPack SDK [8]. The version used in this thesis is JetPack 6.1, which granted a Ubuntu 22.04 with a Linux Kernel (Tegra) 5.15, also known as Jetson Linux 36.3. The whole range of installed packages, can be consulted in here on the official documentation. I chose the last available SDK as it promised the best performances out of the previous iterations, documentation and examples up-to-date.

Setup Jetson

Flashing the NVIDIA Jetson requires booting up the board in recovery mode. To do so, install the NVIDIA SDK Manager on a Ubuntu host, add a jumper among the PINs 9 and 11 of the Jetson, connect the board to the host, and then proceed with the installation [9].

2.3.1 Overall Architecture

The overall system architecture is centered around the NVIDIA Jetson. One or more video cameras can be connected to it, after which the AI model's inference is executed within the DeepStream pipeline. The pipeline is also responsible for transmitting the detection vectors to a server using the communication protocol Kafka. A human operator can access this data through the web application, as detailed in 4.2. The integrated components are summarized in Figure 2.3.

The obtained architecture is modular, as it allows for the replacement of individual components while keeping the rest of the system intact. Other available choices for the robot included the Boston Dynamics SPOT and Unitree's Go2, or any other model as long as the NVIDIA Jetson can be powered. The architecture

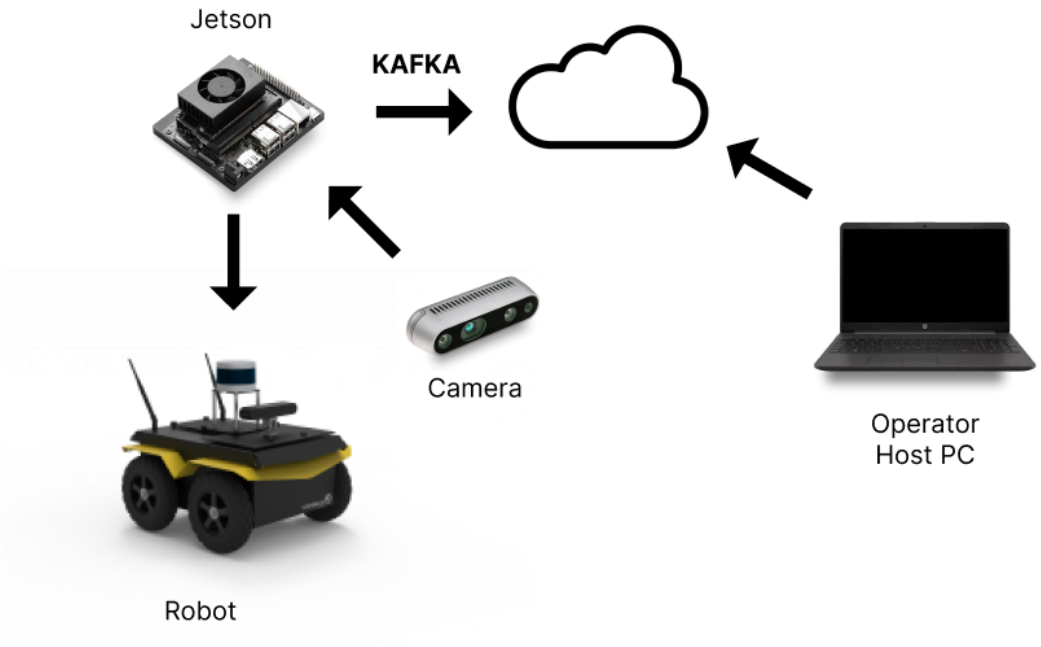


Figure 2.3: System Architecture

is scalable in height, as a more powerful integrated computer can replace the Orin Nano to accommodate a more demanding AI model; it is scalable in width, as exact copies of the same system and hardware can be deployed with the same running codebase and integrated seamlessly. Finally, the system is easy to configure, as it relies on accessible configuration files that enable significant modifications with minimal intervention, even for personnel without coding experience. The mean power consumption is available in Section (5).

Chapter 3

Computer Vision

After defining the idea of Computer Vision and introducing the concept of Neural Networks, this chapter describes the importance of high quality data, the concept of overfitting, how to prepare a dataset and how generative AI can be used to enhance data.

3.1 Introduction to Computer Vision

Computer Vision is an inter-disciplinary field which aims to give computers the ability to acquire, analyze, process and understand data of digital images, in order to produce numerical or symbolic information with the aid of geometry, computer science, statistics, and learning theory. Images can assume various forms, as point clouds from LiDARs, video sequences, 3D scans, 2D images et cetera. The past two decades have seen a massive improvement of feature extraction techniques, largely largely driven by neural networks – computational models loosely inspired by biological brains. These networks, particularly deep learning models like convolutional neural networks (CNNs), have excelled at learning complex, non-linear relationships in data, which are challenging to model using traditional analytical methods. Such machine learning algorithms produce in output a vector of data that can be later reinterpreted. The most common approaches to object detection can be divided into two categories:

1. Single-Shot Detection Methods: These methods skip the region proposal stage and directly predict object classes and bounding boxes in a single step, making them faster and more efficient.

2. Two-Stage Methods: These methods first identify regions of interest (ROIs), or candidate regions—areas that might contain the desired object. These ROIs are then passed into another neural network to classify the object and refine the bounding box. Prominent examples of such networks include R-CNN, Fast R-CNN, and Faster R-CNN.

Two-stage methods tend to be more accurate but are also more computationally intensive, which makes them less suitable for applications requiring low latency, such as the one described in this thesis. For this reason, I utilized YOLO (You Only Look Once), due to its accessibility through the Ultralytics library and its proven performance in real-time applications. The output of this network is a vector that includes the class of the detected object and its bounding box, represented as normalized floats between 0 and 1. A CNN extracts features from the dataset images through a series of convolutional layers, which apply filters to detect patterns such as edges, textures, and shapes. These features are progressively combined to form higher-level representations. The quality of the dataset directly impacts performance; a good dataset is one that is diverse, well-annotated, and representative of the real-world conditions the model will encounter. Diversity ensures the model generalizes well, proper annotations provide accurate learning signals, and representativeness prevents biases that could lead to poor predictions during inference.

3.2 Choosing a model

In the field of object detection, a wide range of models exist, each with its own strengths and trade-offs. Among them, two-stage methods like Faster R-CNN, which was previously discussed in Section 3, have been widely adopted due to their high accuracy. Other alternatives include RetinaNet, which leverages a focal loss function to improve performance on hard-to-detect objects, and SSD (Single Shot MultiBox Detector), known for its balance between speed and precision [10].

A comparative study conducted by researchers in Korea evaluated the performance of RetinaNet-50 and RetinaNet-101 against YOLOv3, concluding that while RetinaNet offered competitive detection capabilities, YOLOv3 significantly outperformed it in terms of inference speed [11]. Similarly, an analysis by Indian researchers referenced multiple object detection architectures, including R-CNN, Fast R-CNN, and Faster R-CNN, highlighting that despite their accuracy, their inference times were prohibitively slow for real-time applications.

Given these considerations, I opted for YOLOv11, as it provides a well-balanced trade-off between speed and accuracy, making it particularly suitable for real-time detection tasks. Additionally, the Ultralyics implementation [12] of YOLO stands out due to its extensive documentation and integrated suite for training, validation, and testing, streamlining the development process.

3.3 Dataset

No publicly available dataset met the specific requirements of this work. The scarcity of suitable datasets for security fence inspection has also been noted in previous studies [13]. To address this gap, I created a custom dataset by integrating web scraping, 3D rendering, and generative AI, each explained in the following chapters. At that stage the model was not generalizing well as expected, as it had a limited size and was missing diverse data, as highlighted in 3.3.1. This lead to the use of `flux.1`, a generative AI model 3.3.1, to generate the needed corner cases. At this point, building the dataset was a continuous iteration that involved the identification of missing data, the addition of freshly generated, classified and labelled data into the dataset, and testing the model. The definitive dataset version contained 4232 images, of which 2320 were anomalies, and the rest was background, as shown in 3.1.

3.3.1 Obtaining the images

Data directly impacts model performance. A diverse and well-varied dataset is essential for ensuring good generalization during inference. Increasing the amount of data in a dataset does not always improve its quality. In fact, it can lead to

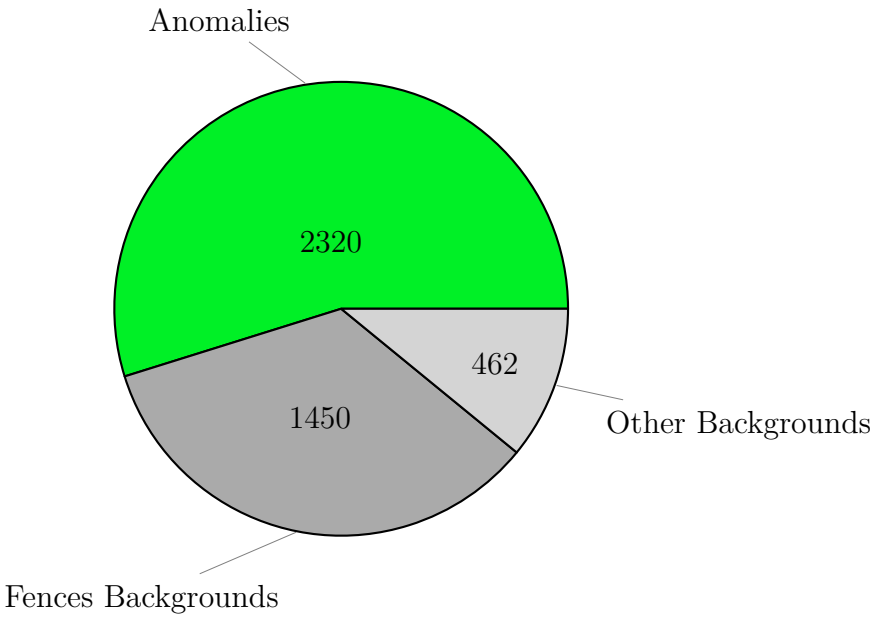


Figure 3.1: Dataset Composition: Anomaly (54.8%), Fences Backgrounds (34.2%), Other Backgrounds (11%)

a condition called *overfitting*, where the model learns the features of the training data so precisely that it struggles to generalize to new, unseen data. One effective approach to enhancing the diversity of a dataset is *data augmentation*, which consists applying transformations to existing data to create variations without collecting additional samples. Common augmentation methods include rotating, flipping, cropping, or altering the brightness, contrast, and colors of images. These manipulations mimic the natural variations seen in real-world scenarios, exposing the model to a broader range of conditions and making it more robust. For example, an augmented dataset might simulate variations in lighting, camera angles, or object positions, helping the model better handle such variations during inference.

Webscraping

The first approach involved web scraping, an action which consists of automatically downloading images from web sources using a Python script. After collecting the images, each image was kept or discarted if they passed these heuristics: a. the image size is below 40KB in size, b. the image is lower than 440px in width or height, c. the image is already present in the dataset.

After this initial filtering, I manually reviewed the dataset, eliminating images that were irrelevant, repetitive, or lacked diversity. This method proved to be time-efficient; however, beyond a certain point, the collected images exhibited significant

redundancy. Even when varying the prompt language, search location, and search engine, the retrieved results remained largely unchanged. More importantly, the number of images containing holed fences was so low that it was statistically insignificant. Nonetheless, web scraping contributed to the diversity of background images, incorporating elements such as intact fences, glass doors, stained glass, and railings. In total, 1300 out of 1912 web-scraped images were classified as backgrounds, representing the majority of background samples in the dataset.

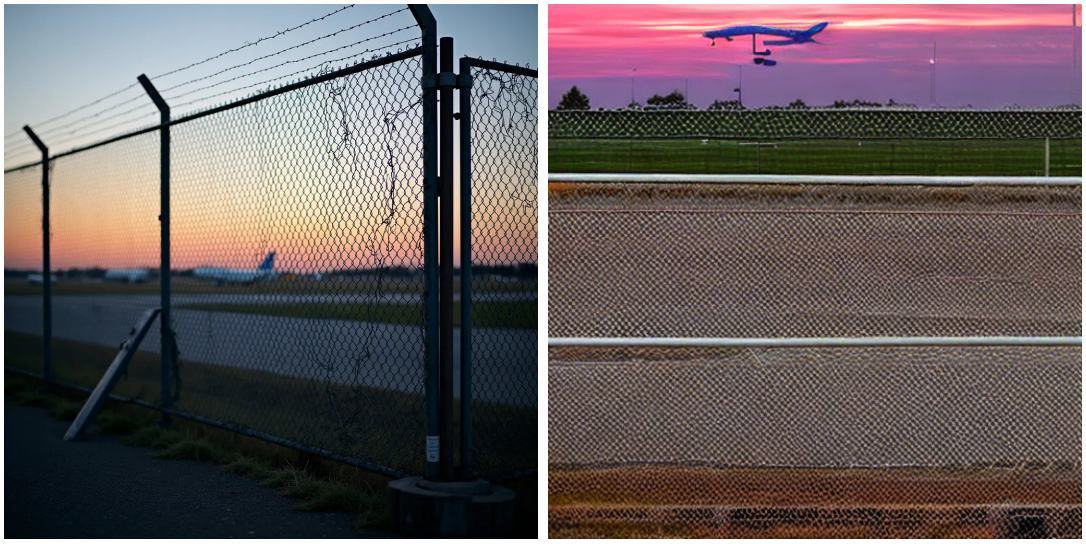
3D Modelling

The second approach involved modeling one or more scenes that encompassed various perspectives, edge-case scenarios, lighting conditions, and obstructions to generate a realistic dataset. While this approach was promising, it proved to be time-consuming, as I had little experience in 3D modeling, despite producing high-quality images. The meshes were freely available online, and my role was limited to setting up the scene, lighting, and camera views. For each scene, I rendered six different frames. The software used was Blender, an open-source tool for 3D modeling. The primary reason this approach was ultimately discontinued was its lack of scalability: while manually modeling each scene for a limited number of classes might have been feasible, it was not a generalizable solution.

Figure 3.2: Rendered images

Generative AI - Flux.1

Despite the data augmentation techniques discussed in Section 3.3, certain types of data remain rare, costly, or impractical to collect. These include extreme edge cases, uncommon object configurations, and scenarios influenced by specific environmental conditions. In such instances, generative networks can be leveraged to synthetically produce these rare cases, enhancing dataset diversity and completeness.



Different models' output given the same prompt[1]. From left to right: `flux.1-schnell`[14], `stable-diffusion-v1.5`[15].

`Flux` is a family of text-to-image models that generates images from natural language prompts, developed by Black Forest Labs. The family employs a hybrid architecture combining multimodal and parallel diffusion transformer blocks, scaled to 12 billion parameters [16]. Flux is available in three different flavours:

- **Schnell**: An open-source variant released under the Apache License
- **Dev**: A source-available version under a non-commercial license.
- **Pro**: A proprietary version accessible via API licensing.

`Flux.1` is a computationally demanding model, which required execution on the workstation described in Section 2.1. Initially, I developed a Python script to perform inference based on a given prompt. However, I later discovered `ComfyUI`, which provided a more efficient and structured approach. After fine-tuning the parameters to achieve an optimal balance between image quality and inference time, I finalized the workflow and parameter configuration presented in the following pages. For the initial prototyping phase, I utilized the `schnell` model due to its faster inference speed. Subsequently, I transitioned to the `dev` model, as it consistently produced objectively superior results under identical conditions. On average, the time required to generate a single image was approximately 12.8 seconds.

ComfyUI works by defining a workflow, a succession of nodes with unique capabilities and parameters. The workflow developed for image generation is represented

in Figure 3.3. The process begins with the `DualCLIPLoader`, which processes the prompt using the models `t5xxl_fp16.safetensors` and `clip_l.safetensors`. The textual input is then encoded through the `CLIPTextEncoderFlux`, where a guidance value of 100.0 ensures that the generated output adheres as closely as possible to the given prompt. The generation starts from random noise, which serves as the initial input for the sampling process. The Euler sampler is employed to refine the image generation, working in conjunction with the `sgm_uniform` scheduler. This scheduler is configured with 50 — in some prompts increased to 75 — and a denoise value of 1.00, ensuring a controlled and effective progression towards the final output. All these parameters are forwarded to the VAE node that contains the flux model, that generates the images in form of tensors that are later converted into a `.png` image.

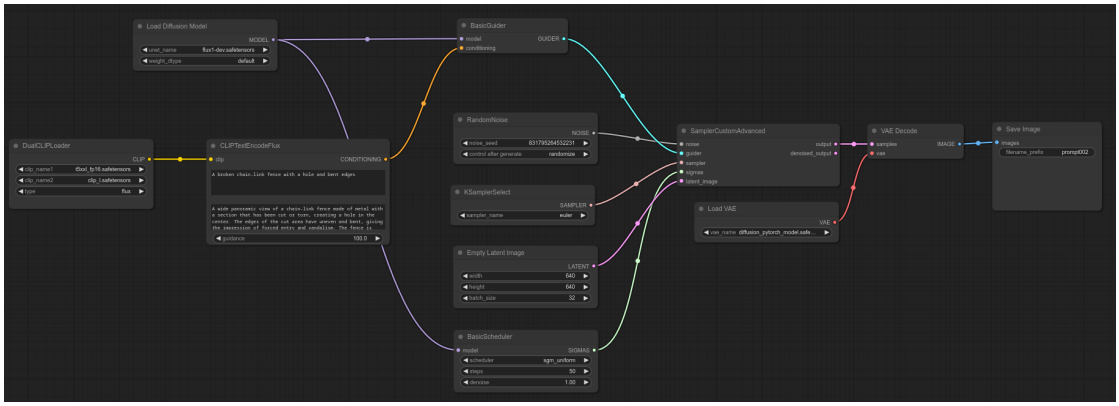


Figure 3.3: The ComfyUI workflow used to generated images.

For each prompt, I generated a images batches and then proceeded to classify and label them as described in 3.3.2. The results are available in Section 3.3.1.

Prompt Engineering

I tested a total of 51 prompts. Each prompt consisted of two inputs: a **CLIP** prompt, a shorter description of, and **T5** prompt, a more complex description that would all the nouances of the scene. If the CLIP input is not given, that field will use the T5 prompt. I did not test what would happen if the CLIP and T5 had two totally different and contrasting description. I generated a number images for each prompt, usually 320, classified them and decided wether to generate more images, or keep editing the prompt until I reached more satisfactory results. This approach lead to the formation of three groups by thematic similarities, described in 3.1

1. **Holes**; The logic applied was to mention the hole, the material, the position, and a bigger hole as I believed it would make them easier to generate. These

Anomaly Background Trash

Figure 3.4: An example of image classification

groups resulted in prompts 015 and 002 yielding the best results due to the use of keywords *pierced, damaged, holed, ruined, showing signs of intrusions, intruders*.

2. **Distance**; The previous prompts generated a good amount of high quality images, that often had a front-faced POV. Since the robot wouldn't walk that close to the fence, the dataset images needed a bigger amount of fences seen from a further distance.

3. **Background complexity**; the previous prompts provided a good amount of data, but when testing the robot on real-case scenarios, the detection were often unsatisfactory, unless the background had a uniform background. To address this issue, I focused on making more complex backgrounds, while keeping a distant, pierced fence, while Reduce the presence of artifact.

	Prompt IDs	**Anomalies**[2]	**Reason**
Group 1	001 → 011	1008	Holed and cut fences
Group 2	015 → 039	750	Distant point of view
Group 3	040 → 051	155	Increase background complexity

Table 3.1: Grouping of prompts based on thematic similarities

[2]Number of images containing at least one anomaly: not all generated pictures had only one anomaly.

Classification Results

Although I tested 51 different prompts, not all of them performed equally. Some generated more backgrounds than anomalies, some were too monotonous, and some didn't generate any meaningful data. A graph showing the percentage of anomalies per-prompt is Figure 3.4. The total amount of generated images was 45425, and among those only 1913 were classified as *anomaly*. More details about each prompt including the percentages of *anomalies* and *backgrounds* is available in Appendix A.

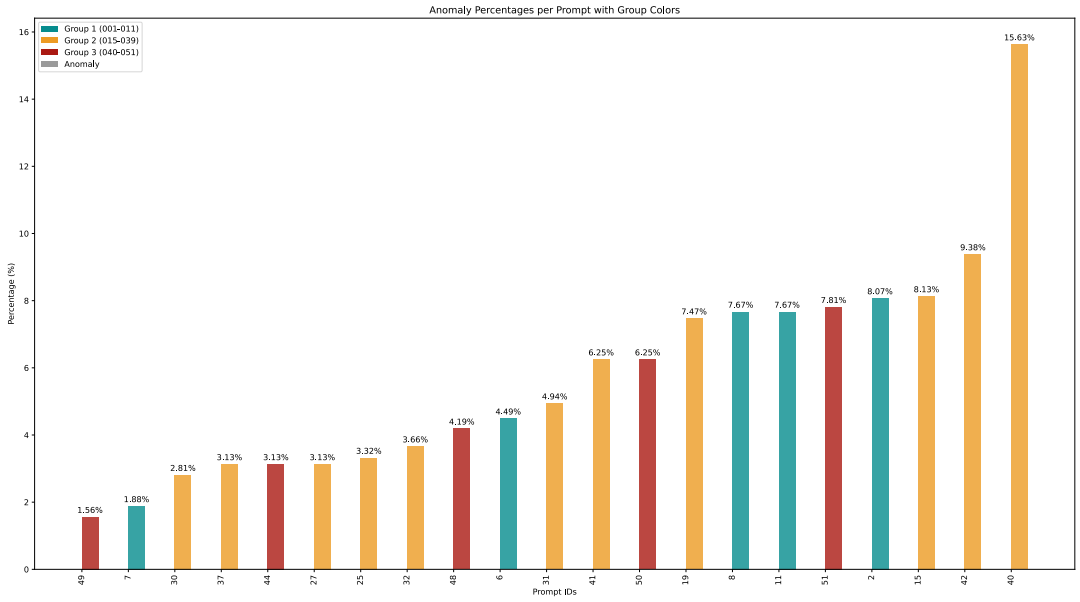


Figure 3.5: Percentage of anomalies over all the generated images, sorted by percentage

3.3.2 Classification and labelling

The images generated by `flux.1` were manually classified as *anomaly* i.e. a damaged fence a *background*, i.e. an image without anomalies corner-case, or *trash*, i.e. images whose generation wasn't real due to the presence of artifacts or unrealistic composition, materials, or unnatural lighting. An example of classification is available at 3.4. The classification happened with a software I developed, Sorter, which is freely available on GitHub [17]. Once all the images was classified, I manually labelled the *anomaly* class with label-studio [18]. Once the images were classified I labelled them using the FOSS software Label Studio [18] to help assigning the bounding boxes to the anomalies. The overall amount of time.

Figure 3.6: Sample anomalies from different prompt outputs. First row: group 1, second row: group 2, third row: group 3

Chapter 4

DeepStream Pipeline

A pipeline is the component responsable for acquiring a source, running the inference of the AI model, generate the detection vectors and finally send it with broker. This chapter explains in detail the plugins into play, and the different configurations developed for the tests.

4.1 Pipeline Structure

The pipeline is a tree of *plugins*, each receiving the output of the previous one, except for the source plugin and other special cases. Data flows between plugins through `caps` and `capsfilter`, which define buffer capabilities to ensure desired properties before passing data forward.

Caps can be automatically negotiated [19], or be manually set when constructing a pipeline from scratch. A `capsfilter` defines the media type and format of data within a pipeline, restricting the types of caps allowed. It ensures that only specific formats, such as `YUV`, `RGB`, or `H.264`, are used before reaching a decoder or processing element, making it crucial when setting media formats after a source element like a camera or file source. A `pad` connects elements, allowing data flow between them. Elements communicate by linking their pads, where source pads output data and sink pads receive it. While pads exist throughout GStreamer, they are explicitly visible in certain elements like `nvv4l2decoder` and `nvstreammux`, due to how GStreamer handles linking and dynamic elements. Pads can be static, remaining unchanged throughout an element's lifetime, or dynamic (request pads), created only when needed. Here is an explaination of the plugins used in the various pipelines 4.1.1

- **Gst-v4l2src** Camera source, raw format depend on the device used.
- **Gst-urisourcebin** URI source, usually a `.h264` or `.h265` video
- **Gst-h264parser** parses the source video into a raw format that has to be converted by `Gst-vidconv` or `Gst-nvvidconv`
- **Gst-vidconv** and **Gst-nvvidconv**. It's used to ensure that a set of raw formats are supported — including the `YUV2` colour space used by the Intel RealSense2.1.1 — and are converted into a format readable by the following plugins; in particular by converting the video feed into NV12 buffers that are readable by the the `streammux` plugin. In case we have a camera with raw format in `nvvideoconvert`, GStreamer plugins' capability negotiation shall be intelligent enough to reduce compute by `vidconv` doing passthrough [19]
- **Gst-nvstreammux**, it forms a batch of frames from one or multiple input sources, and forward them to the inference plugin.
- **Gst-nvinfer**, inference engine, often called *PGIE*, runs the inference over the input batches and returns metadata that includes the eventually found bounding boxes.

- **Gst-nvtracker**, allows the use of a low-level tracker library to track the detected objects over time persistently with unique IDs, gets as input the output of `Gst-nvinfer`

- **Gst-tee**, a plugin that forwardes an input over two branches, see Figure 4.5 for an example.

- **Gst-nvdsosd** it draws bounding boxes, text, arrows, lines, circles and region of interest (RoI) polygons.

- **Gst-msgconv** converts metadata into a message payload based on a schema. Its output is the input `Gst-nvmsgbroker` plugin.

- **Gst-msgbroker** a plugin that sends payload messages to the server using a specified communication protocol. The out-of-the-box supported protocols are: Kafka, Azure IOT, AMQP, REDIS, MQTT. Moreover, DeepStream allows its support through a custom implementation of the adapter interface.

These elements can be combined to produce a working pipeline that extracts the raw buffers from the video feed, runs the inference, and shows the results on screen and/or sends them with a communication protocol. The NVIDIA DeepStream SDK offers multiple ways for creating such pipeline, described in the subsections below.

Configuration file

The simplest way to generate a pipeline by setting up a `.txt` configuration file that can be run with `deepstream-app -c config.txt`. The `deepstream-app` command invokes a C code that reads the configuration file and dynamically generates a pipeline applying the specified configurations listed in [20]. This offers a fast way to prototype pipelines, and I used them extensively for testing reasons. However, this approach leaves no space for accessing the pipeline data, which excludes — among many other possibilities — the chance of sending images. An example configuration file is represented in 4.1.

```
[application]
enable-perf-measurement=1
perf-measurement-interval-sec=5

[source0]
enable=0
type=1
camera-width=640
camera-height=480
camera-fps-n=30
camera-fps-d=1
camera-v4l2-dev-node=4

[streammux]
gpu-id=0
live-source=1
batch-size=1
batched-push-timeout=40000
width=640
height=480
nvbuf-memory-type=0

[osd]
enable=1
gpu-id=0
text-size=8
text-color=1;1;1;1
text-bg-color=0.4;0.4;0.4;0.4
font=Serif
nvbuf-memory-type=0

[primary-gie]
enable=1
model-engine-file=../models/detector_11m_v6_fp16.engine
config-file../src/testing_pipeline_cfg_pgie.txt

[sink0]
enable=1
type=222
sync=0
width=640
height=480
plane-id=1
source-id=0
```

Table 4.1: Example pipeline, equivalent to the one simplified camera pipeline described in 4.2

Writing a pipeline in C

To solve that problem and to have a better understanding of the data flow and the variables involved, I had to code either in C or Python the pipeline. Among the two choices, I chose C as it is the officially supported language, and DeepStream docs uses C examples, instead of the Python wrapping. In contrast to the configuration file, it is much harder to maintain, requires manual intervation for setting caps correctly.

4.1.1 Pipelines

The next sections will show the pipelines used for testing and for deployment. I grouped by color the plugins by their functionality. The green plugins are used for getting the video input, the orange plugin for the artificial intelligence model, the blue are for communication, the pink for showing the output to screen (on screen device), and fiannally the grey are uncategorized.

Simplified URI Pipeline

Figure 4.1: Simplified URI Pipeline, a minimal pipeline using a URI source

The most basic pipeline used throughout the thesis, it takes a video in format `.h264`, and runs the inference on it.

Simplified Video Source Pipeline

Figure 4.2: Simplified Video Source Pipeline, a minimal pipeline using a camera video feed

Same as 4.1, but it takes a camera video feed.

Simplified Double Video Source Pipeline

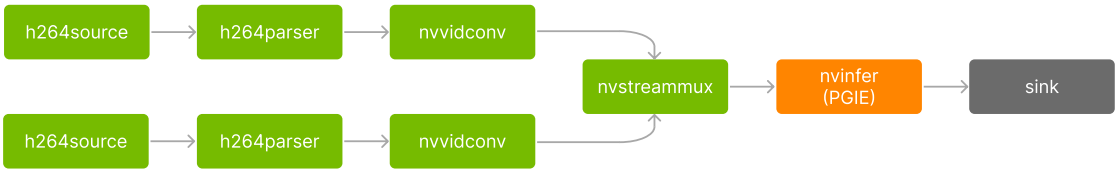


Figure 4.3: Pipeline used for testing the loss of performances with three camera feeds

Same as 4.2, but the two branches allow the stream of two different cameras

Simplified Triple Video Source Pipeline

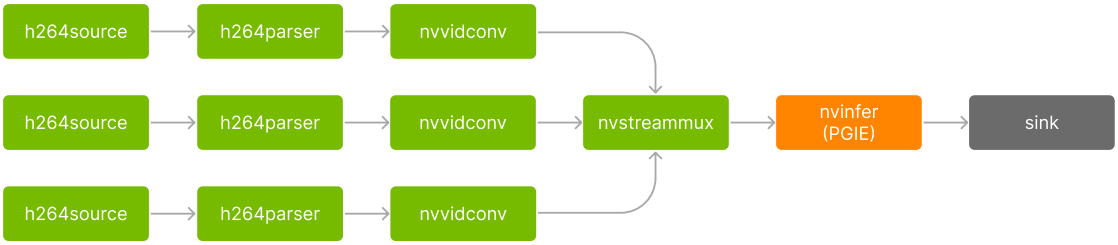


Figure 4.4: Pipeline used for testing the loss of performances with three camera feeds

Same as 4.2, but the two branches allow the stream of three different cameras

Debuggable Patrolling Pipeline

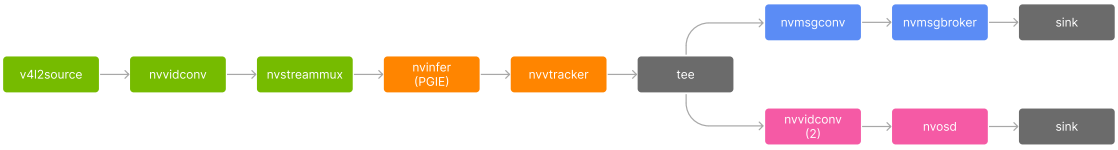


Figure 4.5: Pipeline used for testing before deployment, featuring a tracker, a communication broker, and video-screen output

It features a single video feed, a tracker to assign unique IDs to the detected objects, and then the data is forwarded in two directions, in one it gets converted to a message format and sent through the Kafka communication protocol; while the other branch shows the camera feed and the detections on screen. Unlike the

previous ones, this one wasn't dynamically generated, as doing so wouldn't allow for encapsulating custom data, such as the encoded image, in the message.

Patrolling Pipeline

Figure 4.6: Pipeline used during deployment, featuring a tracker and a communication broker

Identically to the previous one, but the branch containing the OSD has been disabled.

TensorRT Engine

In DeepStream, the `.engine` file format is a serialized TensorRT model optimized for deployment on NVIDIA GPUs. This format allows for efficient inference by leveraging hardware acceleration and reducing the overhead associated with model execution. The `.engine` file must be generated from a compatible model format, such as `.onnx`, before it can be used within a DeepStream pipeline. To convert the `.pt` model into `.onnx` format, I relied on the method provided by MarcosLuciano [**repo:marcos-luciano**]. Once the `.onnx` file is obtained, it can be passed to the `model-onnx-file` field within the DeepStream configuration. When executed, the pipeline automatically generates the corresponding `.engine` file, making the process seamless. An alternative approach involved using the `trtexec` binary located at `/opt/nvidia/bin/tensorrt/trtexec`, which allows for direct conversion of `.onnx` models to TensorRT format. However, despite experimenting with different parameter configurations — including specifying input size and tensor input/output names — I was unable to produce a functional model. The only method that consistently worked was the one provided by MarcosLuciano's repository, which also supplied the necessary library for custom YOLO inference. Without this library, running the model within DeepStream would not have been possible.

4.2 Server e Endpoint

DeepStream provides a message broker plugin that natively supports Kafka, as described in Section 4.1. For the purposes of this thesis, the Jetson itself was used as the Kafka and Zookeper servers. However, this approach introduces unnecessary computational overhead and increases the Jetson's power consumption and CPU usage. Ideally, the server should be a separate machine to optimize efficiency.

```
{
  "version" : "4.0",
  "id" : "390",
  "@timestamp" : "2025-01-22T15:09:58.929Z",
  "sensorId" : "CAMERA_ID",
  "objects" : [
    "4|102.319|336.987|192.717|446.415|broken fence",
    "5|496.585|77.7915|599.111|173.693|broken fence",
    "6|94.3388|70.4204|210.873|197.156|broken fence",
    "7|269.562|65.048|394.132|209.711|broken fence",
    "8|261.067|308.172|396.738|474.889|broken fence"
  ],
  "customMessage" : [
    "image;jpg;1920.0x1080.0;2025-01-22T15:09:58.924Z;9j/
    4AAQSkZJRgABAQAAAQABAAD/2wBDAAMCAgMCAgMDAwMEAwMEBQgFB
    QQEBQoHBw..."
  ]
}
```


Table 4.2: Example of a sent Kafka message. The `customMessage` field has been trimmed down for this representation. Originally it was 588.476 characters long. The objects array contain the unique ID attached by the tracker, the bounding box coordinates, and the class name

The detection module is capable of functioning without an active server, but in such cases, all unsent data is lost, as it is not stored persistently. This limitation is one of the issues highlighted in Section 6. Assuming the server is operational, the Jetson transmits a message once per second, even in the absence of detections. In such instances, the messages contain mostly empty fields, with the exception of those shown in Figure 4.2. The image data is stored in the `customMessage` field as a Base64-encoded string and transmitted to the server. Upon receiving a message, the system searches for the `customMessage` field. If it is absent, the message is discarded. Otherwise, the encoded image is extracted, decoded, and stored locally for easy access. The operator can then access the web application, which automatically retrieves and displays the image along with bounding boxes and associated metadata. Additionally, the interface provides access to previous detections for further analysis.

Chapter 5

Results

Results concerning the components of the system. I will analyze the Neural Network performances, considering various setups with one, two and three cameras, as well as the energy, memory and video memory consumption, while adhering to the system constraints.

5.1 Model Performance

These experiments were conducted using the `.pt` model file on the laptop described in the Hardware section (2.1), rather than directly on the Jetson device. This decision was made due to the constraints imposed by DeepStream, which does not provide a straightforward method for extracting detection data in a structured manner. The only viable alternative would have been to develop a custom pipeline from scratch, incorporating only the essential plugins needed for inference and implementing a custom data retrieval algorithm that would have lowered the performances. However, as discussed in previous sections, designing such a pipeline is a time-consuming process that requires extensive configuration and debugging. Given the scope and time constraints of this thesis, prioritizing model evaluation on a more flexible development environment was the most practical choice. The accuracy results obtained from these tests are presented in Figure 5.1.

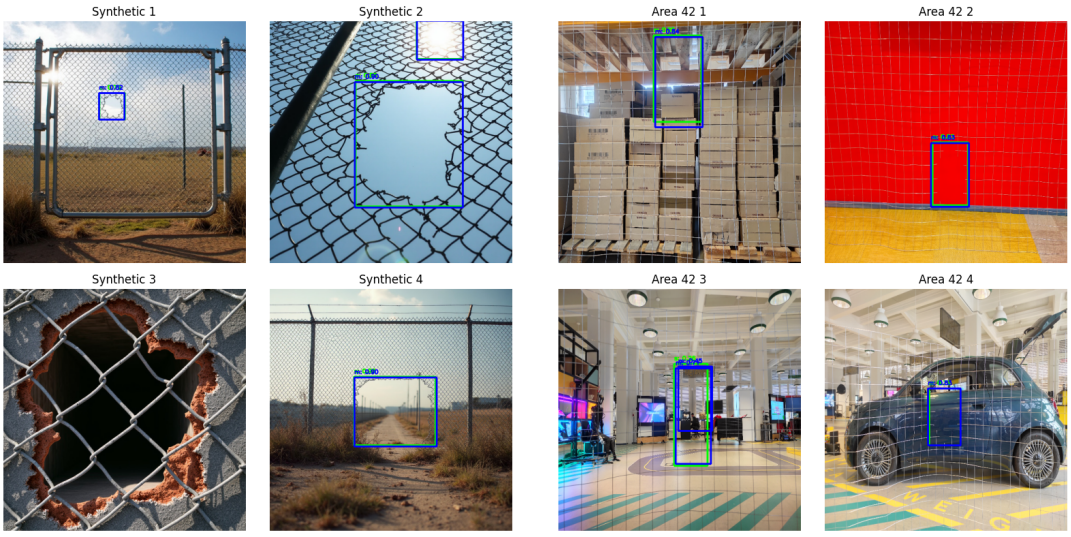


Figure 5.1: Inference results for three different models n(orange), s(green), m(blue). The models n and s show identical accuracies and bounding boxes.

For testing purposes, I selected four synthetically generated images and four real-world examples, each containing a single object to be detected. The corresponding inference results are presented in Figures 5.1 and 5.2. In some cases, the models detected multiple objects within the same image; however, any additional detections consistently exhibited confidence scores below 0.45. In a real-world deployment scenario, such low-confidence detections would typically be discarded through thresholding to ensure only reliable inferences are considered.

Notably, models s and n produced almost identical accuracy values across all test

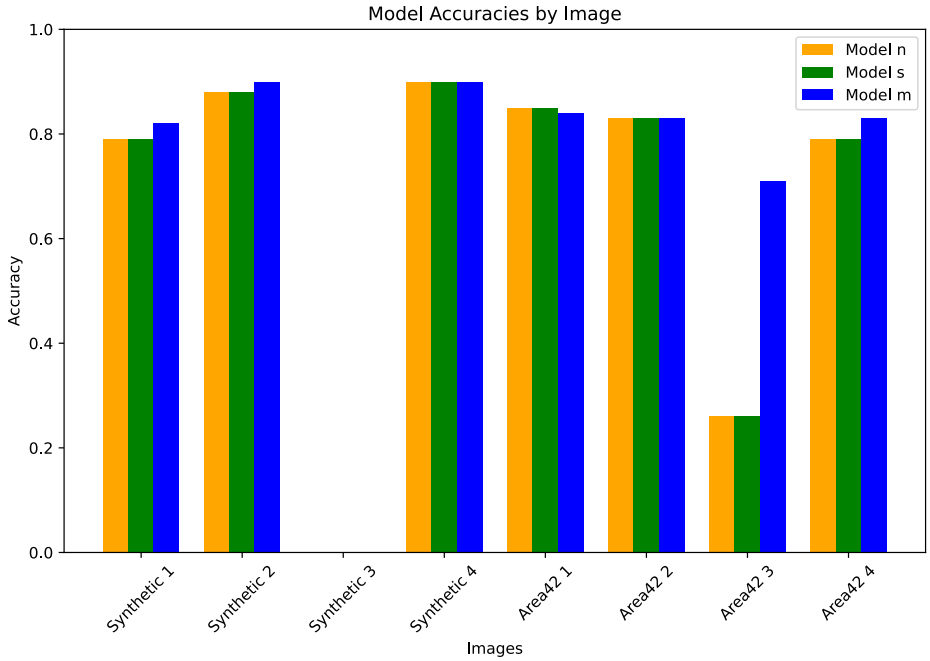


Figure 5.2: Accuracies for each model grouped by image.

images, despite having different parameter counts. This behavior is unexpected, as a higher number of parameters generally correlates with improved representational capacity and inference performance. The reason for this phenomenon remains unclear and may warrant further investigation. In contrast, model m consistently outperformed both s and n across all test cases, as anticipated. This result aligns with expectations, given that model m is designed with a larger capacity and greater computational complexity, which typically enables more precise object detection.

5.2 Jetson

The goal of this section is to determine the optimal model size for inference on the Jetson platform, considering hardware constraints and the specific use case.

This analysis compares different model sizes in terms of power consumption, RAM usage, CPU and GPU load. The Jetson can operate in two power modes: 15W and 7W. In the lower power mode, the system engages power efficiency optimizations to extend operational time. Moreover, RAM usage must remain below 8GB. The Neural Network training times are negligible due to minimal variation. Power consumption varies depending on the robot's configuration and

workload. It is important to note that the recorded consumption values are expected to be higher than those observed in real-world deployment due to the active Ubuntu X11 session, required for running the experiments, and by the polling active every second, introduced for collecting data.

The models are nano(n), small(s), medium(m). Each one of them has been either not quantized (FP32), or quantized to FP16. This leaves us with 6 models to compare for a given dataset. All results are compared against an idle baseline to ensure a clear understanding of additional resource usage introduced by model inference, and evaluated using the same video feed to ensure replicability.

5.2.1 Results and Analysis

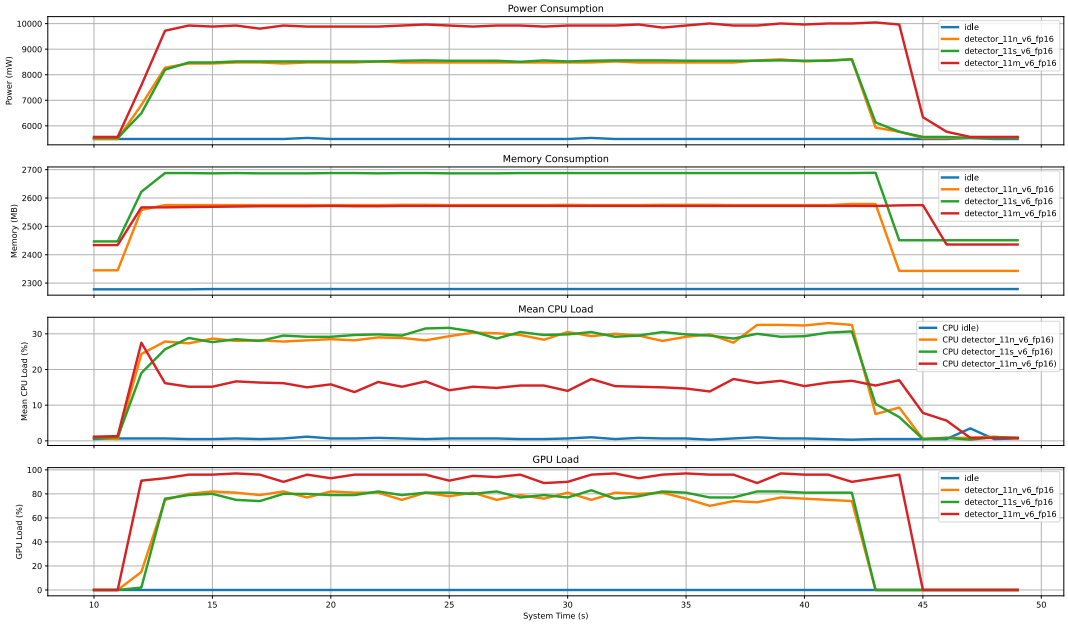


Figure 5.3: Comparing three different models trained under the same dataset

As expected, smaller models tend to consume less energy than larger models. However, an unexpected observation is that the small and nano models exhibit nearly identical power consumption. Similar trends happen for the FP32 model.

- FP16 quantized models exhibit, on average, a 5% increase in CPU load.
- FP16 nano and small models consume less power than any FP32 model.
- FP16 medium consumes power at levels comparable to all FP32 models

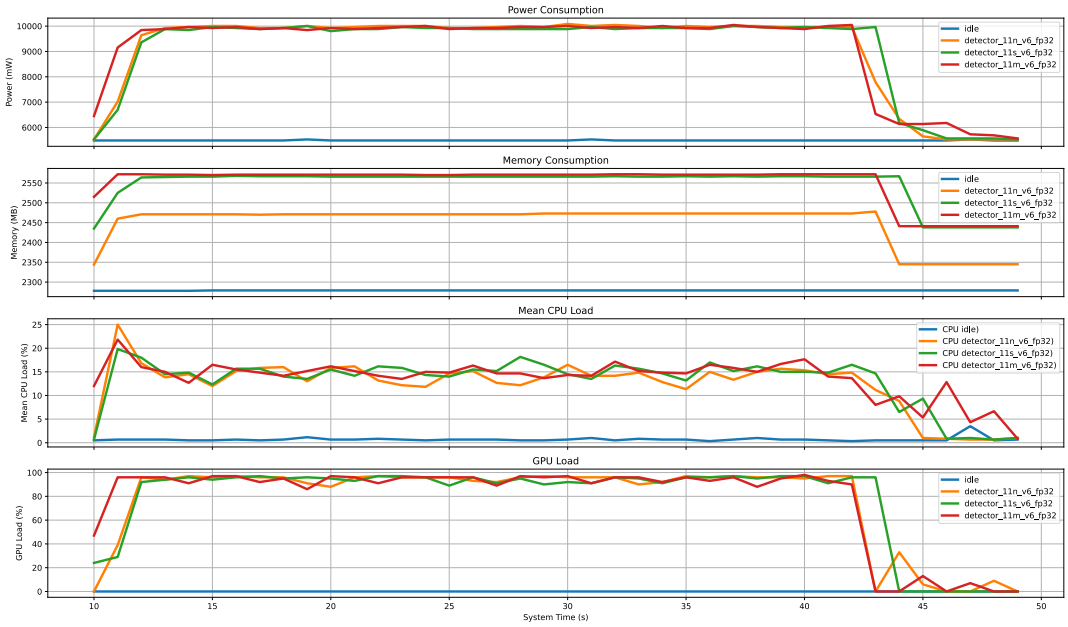


Figure 5.4: Pipeline used during deployment, featuring a tracker and a communication broker

- Memory usage, GPU load, and CPU load remain unaffected by an increased number of detections

These findings highlight the importance of selecting an appropriate model size based on power and efficiency requirements

5.2.2 Cameras

It is expected that as the number of cameras in the system increases, the pipeline's performance will degrade due to the increased computational load required to process multiple images simultaneously—determined by the batch-size parameter. To assess this impact, the largest model without quantization (FP32) was selected as a reference, and FPS values were collected through polling. The pipeline configuration for two cameras is shown in Figure 4.3, while the configuration for three cameras is depicted in Figure 4.4.

Data was collected for the following configurations, both with two and three cameras, for each of the three models (n, s, m):

- 640x480, 60 FPS
- 960x540, 30 FPS

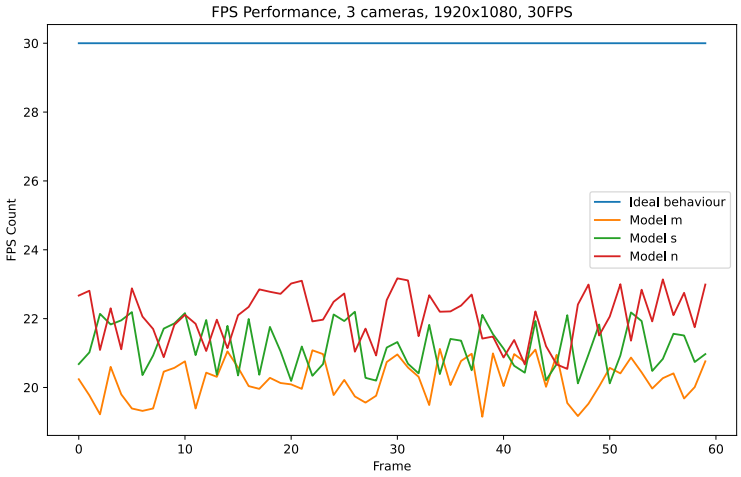


Figure 5.5: FPS performances using three cameras, a FHD resolution capped at 30FPS.

- 1920x1080, 30 FPS

Each configuration successfully maintained the expected FPS. The only exception, and the most notable case, was the setup with three cameras—comprising two Intel RealSense D345i cameras connected via USB 3.1 and one via USB 2.0—which exhibited significant performance drops across all models. The corresponding results are illustrated in Figure 5.5.

Chapter 6

Conclusions and Future works

The camera utilized in this thesis has not been leveraged to its fullest potential, as only the YUV color space was explored, leaving other features, such as the infrared (IR) capabilities, unutilized. Incorporating IR functionality could significantly enhance the system's performance in low-light or nighttime conditions, where standard visible light imaging often fails. This improvement would expand the system's operational scenarios, enabling effective 24/7 surveillance in outdoor environments.

Due to unforeseen challenges in development and semantic ambiguities, the detection of intruders and PPE compliance had to be excluded from the system. Ideally, the implementation would have leveraged DeepStream's architecture, employing a primary GPU inference engine (PGIE) to detect humans and multiple secondary GPU inference engines (SGIEs) to analyze PPE compliance. During working hours, the PGIE would identify human presence, while SGIEs would determine whether individuals were wearing the required PPE. Any person without PPE would be flagged as suspicious, requiring human operator assessment. Outside of working hours, any detected person would automatically be classified as an intruder. However, this approach presents a fundamental semantic issue: how should an intruder be defined? The initial solution—classifying anyone without PPE as suspicious—was a convenient but flawed implementation. It is easily circumvented and does not account for contextual nuances, ultimately rendering it an unreliable criterion for intrusion detection.

A critical area for improvement is the interaction between the Jetson module, a central server, and the web application. The current architecture assumes a reliable network connection between these components. However, in practical deployments, network instability or disconnections could impact real-time anomaly

detection and response. Future implementations should consider optimizing data synchronization mechanisms between the Jetson and the central server to mitigate potential latency issues. Moreover, the integration of the web application with the central server should be refined to ensure that real-time alerts and anomaly reports are consistently delivered, regardless of connectivity interruptions.

Additionally, the use of OpenRMF (Open Robotics Middleware Framework) presents an opportunity for more efficient robot fleet coordination and task allocation. Integrating OpenRMF into the system could improve multi-robot collaboration, path planning, and scheduling, further enhancing the scalability of security operations. This would enable a more adaptive patrolling system where multiple robots operate in sync while minimizing redundant coverage.

The current patrolling algorithm is a primitive point-to-point navigation strategy that relies solely on the robot's perception and local planning. While obstacle avoidance is handled by the robot's onboard systems, the implementation does not incorporate higher-level path optimization strategies. Future work should focus on developing a more advanced patrolling algorithm that incorporates predictive path planning, dynamic obstacle avoidance, and adaptive route adjustments based on detected anomalies.

Moreover, the module relies entirely on the robot's self-awareness regarding battery status and operational constraints. This assumption limits proactive energy management strategies that could optimize mission duration and minimize downtime. Future enhancements should explore predictive battery monitoring and automated return-to-base protocols to prevent mission failures due to power depletion. The system also assumes that the robot remains continuously connected to the internet, with the Jetson module connected to the same network. The current implementation relies on Ubuntu's Wi-Fi auto-connection feature, which, while generally reliable, introduces a vulnerability: should connection errors occur, such as expired credentials, manual intervention is required. This limitation highlights the need for an automatic fallback mechanism, such as mobile network failover or periodic reconnection attempts, to ensure uninterrupted system operation. Another limitation lies in the system's response when an anomaly is detected. Currently, if the robot encounters difficulty in identifying a particular anomaly, it does not attempt to move closer for further investigation. This limitation stems from the lack of integration between the Jetson's detection module and the robot's movement system. A more sophisticated implementation should enable dynamic anomaly investigation, where the robot autonomously repositions itself to obtain a better perspective when detection confidence is low. This would significantly improve anomaly verification and reduce false positives.

Addressing these challenges and incorporating these improvements would enhance the system's robustness, adaptability, and real-world applicability, ultimately making it a more effective solution for autonomous security monitoring.

List of Tables

List of Figures

Appendix A

Prompts

This appendix shows the prompts used to generate images as explained in Section 3.3.1. Each table below lists in order the prompt ID, the total amount of images generated **T**, the number of images classified as anomalies **A**, the number of images classified as backgrounds **B**, the **T5** and **CLIP** prompts, and an eventual **comment** on their results.

ID 2	**T** 2616	**A** 211 (8.07%)	**B** 110 (4.20%)

T5: A wide panoramic view of a chain-link fence made of metal with a section that has been cut or torn, creating a hole in the center. The edges of the cut area have uneven and bent, giving the impression of forced entry and vandalism. The fence is held up by sturdy metal posts, and there is a latch mechanism visible near the top, suggesting it may be part of a gate. The chain-link pattern consists of diamond-shaped openings created by interwoven metal wires.
CLIP: A broken chain-link fence with a hole and bent edges

Comment: Monotonous and close POV

ID 3	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A wide panoramic view of a chain-link fence made of metal with several sections that have been cut or torn, creating multiple holes of varying sizes. The edges of the cut areas are uneven and bent, suggesting forced entry and vandalism. The fence is held up by sturdy metal posts, and there is a latch mechanism visible near the top, indicating it may be part of a gate. The chain-link pattern consists of diamond-shaped openings created by interwoven metal wires
CLIP: A broken chain-link fence with multiple holes and bent edges

ID 4	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A wide panoramic view of a chain-link fence made of metal with a section that has been cut or torn, creating a hole. The edges of the cut area have uneven and bent, giving the impression of forced entry and vandalism. The fence is held up by sturdy metal posts, and there is a latch mechanism visible near the top, suggesting it may be part of a gate.
CLIP: A vandalized and broken chain-link fence with a hole

ID 5	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A wide panoramic view of a chain-link fence made of metal with a section that has been cut or torn, creating a hole. The edges of the cut area are clean, uneven and bent, giving the impression of a forced entry and vandalism. The fence is held up by sturdy metal posts.
CLIP: A vandalized and broken chain-link fence with a hole

ID 6	**T** 3097	**A** 139 (4.49%)	**B** 188 (6.07%)

T5: A wide panoramic view of a chain-link fence made of metal with a section that has been cut or torn, creating a hole. The edges of the cut area are clean, uneven and bent, giving the impression of a forced entry and vandalism, but without a sign of corrosion. The fence is held up by sturdy metal posts.
CLIP: A vandalized and broken chain-link fence with a hole

Comment: Better than 002, diversified POV, sometimes unnecessairly close, overall not too satisfying

ID 7	**T** 1600	**A** 30 (1.88%)	**B** 17 (1.06%)

T5: A wide panoramic view of a chain-link fence made of metal with a section that has been cut or torn, creating two holes. The edges of the cut area are clean, uneven and bent, giving the impression of a forced entry and vandalism. The fence is held up by sturdy metal posts.
CLIP: A vandalized and broken chain-link fence with two holes

Comment: Similar to 002

ID 8	**T** 4096	**A** 314 (7.67%)	**B** 215 (5.25%)

T5: A bottom up view of a chain-link fence made of metal with a section that has been cut or torn, creating a hole. The edges of the cut area are clean, uneven and bent, giving the impression of a forced entry and vandalism, but without a sign of corrosion. The fence is held up by sturdy metal posts.
CLIP: A vandalized and broken chain-link fence with a hole

Comment: Bottom-up POVs, often too close, sometimes it breaks the wall behind without touching the fence, making an interesting corner-case for backgrounds

ID 11	**T** 4096	**A** 314 (7.67%)	**B** 215 (5.25%)

T5: A detailed wide panoramic view of a chain-link fence in an outdoor setting, showcasing significant wear and damage. The fence features bent and frayed wires, with areas of rust highlighting its age and exposure to the elements. The surrounding environment appears subdued, with muted tones suggesting overcast lighting or an abandoned area. The composition focuses on the texture and decay of the fence, emphasizing the rugged, weathered look of the materials and the overall sense of neglect in the scene
CLIP: A weathered chain-link fence with visible damage, bent wires, rusted edges, and an outdoor setting with muted tones

Comment: Mostly monotonous backgrounds

ID 15	**T** 320	**A** 26 (8.13%)	**B** 7 (2.19%)

T5: A wide-angle view of a chain-link fence stretching across the frame, made of metal with visible damage where a section has been cut or torn, forming an irregular hole. The surrounding environment appears expansive and open, and the fence is supported by sturdy posts, with the damaged area being a focal point.
CLIP: A wide view of a vandalized chain-link fence with a large hole.

Comment: Nice POVs but monotonous pictures

ID 16	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A distant, panoramic view of a chain-link fence that has a large, irregular hole. The metal wires around the hole are bent and uneven, suggesting deliberate damage. The fence is framed by sturdy posts, and the background reveals a broad, outdoor environment.
CLIP: A distant perspective of a damaged chain-link fence spanning the frame.

Comment: Mostly trash, front-faced POV. Unlike previous iterations, less trash

ID 17	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A far-reaching view of a metal chain-link fence spanning across the landscape, showing a prominent hole where the wires have been forcibly cut or torn. The clean but uneven edges of the damage are striking, while the surrounding area is characterized by open skies and muted tones.
CLIP: A broad view of a chain-link fence with a torn section and bent wires.

Comment: tutto immondizia, pov dritto in faccia. A diff di prompt016 l'immondizia è più "piccola"

ID 18	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A distant view of a chain-link fence, showcasing its weathered appearance and a large torn section. The hole has jagged, bent edges, with the surrounding area offering a wide perspective of an outdoor setting.
CLIP: A wide-angle shot of a vandalized chain-link fence with visible damage.

Comment: Similar to 017

ID 19	**T** 3520	**A** 263 (7.47%)	**B** 7 (0.20%)

T5: A wide-angle scene of a chain-link fence with a prominent hole near the center. The fence extends far into the distance, emphasizing its scale, and the damaged section shows bent and uneven edges, suggesting recent vandalism.
CLIP: A faraway perspective of a chain-link fence with a large, irregular hole.

Comment: Interesting distant POVs, less trash than usual, amazing backgrounds

ID 20	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A far and distant view of a chain-link fence with visible damage, including a large, irregular hole. The bent and uneven wires contrast with the sturdy metal posts holding up the structure, against an expansive outdoor setting.
CLIP: A faraway perspective of a chain-link fence with a large, irregular hole

ID 21	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A far and distant view of a chain-link fence with visible damage, including a large, irregular hole. The bent and uneven wires contrast with the sturdy metal posts holding up the structure, against an expansive outdoor setting
CLIP: A distant scene of a chain-link fence with torn wires and a wide environment.

Comment: All monotonous backgrounds, no trash but no holes either...

ID 22	**T** 320	**A** 3 (0.94%)	**B** 1 (0.31%)

T5: A panoramic outdoor view of a metal chain-link fence, featuring a significant tear or hole. The wide perspective highlights the length of the fence, with its damaged area adding visual interest in the open space.
CLIP: A chain-link fence with a noticeable hole, viewed from a wide angle.

Comment: Mostly close front-faced POV

ID 23	**T** 320	**A** 3 (0.94%)	**B** 0 (0.00%)

T5: An outdoor setting with a chain-link fence stretching far across the frame, marked by a torn section with bent and uneven wires. The expansive view emphasizes the surrounding environment as much as the damaged structure.
CLIP: A far, outdoor view of a vandalized chain-link fence with a prominent tear

Comment: Same as 022

ID 24	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A wide, distant view of a vandalized chain-link fence, highlighting a torn hole with frayed and bent metal wires. The scene emphasizes the expanse of the fence and its integration into the open outdoor landscape.
CLIP: A wide perspective of a broken chain-link fence in an expansive outdoor setting

Comment: Same as 022

ID 25	**T** 2080	**A** 69 (3.32%)	**B** 0 (0.00%)

T5: A sweeping view of a long chain-link fence extending across an open outdoor environment, with a torn section creating an irregular and jagged hole. The damage appears deliberate, with bent and uneven wires forming sharp edges, while the surrounding scene emphasizes the broad, expansive landscape under muted light. The sturdy metal posts hold up the fence, contrasting with the fragile, damaged section that serves as the focal point
CLIP: A wide view of a damaged chain-link fence with a jagged hole in an open outdoor setting.

Comment: Diversified, distant front POV, with some bottom-up POVs. Some trash but less than usual, the background is always cloudy

ID 26	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A broad view of a weathered chain-link fence stretching across the landscape, featuring a prominent, vandalized section where a large hole disrupts the uniformity of the wires. The torn edges are bent and uneven, but free of rust, suggesting recent damage. The setting is expansive, with the fence dividing the open space and standing out as the primary feature of the composition
CLIP: A panoramic view of a broken chain-link fence dividing an expansive outdoor area

Comment: Nice front POV, but mostly trash

ID 27	**T** 640	**A** 20 (3.13%)	**B** 0 (0.00%)

T5: A front-facing, wide-angle view of a chain-link fence stretching horizontally across the scene, made of metal and featuring a visible hole at its center. The torn section is irregular, with bent and uneven wires suggesting intentional damage. The surrounding environment is open and expansive, emphasizing the rugged texture of the metal fence against a muted, outdoor background. The fence is supported by sturdy vertical posts, which frame the damaged section clearly.
CLIP: A front-facing, wide view of a damaged chain-link fence with a torn hole in an expansive outdoor setting.

Comment: Similar to 026

ID 28	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A frontal, panoramic view of a long chain-link fence spanning across the frame, with a large, jagged hole interrupting its structure near the center. The damaged wires are bent outward, forming sharp and uneven edges that stand out against the clean, sturdy metal posts supporting the fence. The surrounding area is vast and open, with the wide composition emphasizing the fence as the focal point.
CLIP: A front-facing, panoramic view of a broken chain-link fence with a prominent hole in an open area.

ID 29	**T** 320	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A straight-on, wide-angle view of a chain-link fence extending across the entire scene. The fence features a large, irregular hole near its center, with frayed wires bent and curling outward, suggesting deliberate vandalism. The sturdy metal posts frame the damaged section, while the expansive, outdoor setting highlights the scale and structure of the fence against the muted tones of the background.
CLIP: A wide, front-facing view of a chain-link fence with a large torn hole in an expansive outdoor setting.

ID 30	**T** 3200	**A** 90 (2.81%)	**B** 0 (0.00%)

T5: A front-facing wide-angle view of a chain-link fence stretching across the frame, made of metal with visible damage where a section has been cut or torn, forming an irregular hole. The surrounding environment appears expansive and open, and the fence is supported by sturdy posts, with the damaged area being a focal point.
CLIP: A front-facing wide view of a vandalized chain-link fence with a large hole.

ID 31	**T** 3200	**A** 158 (4.94%)	**B** 0 (0.00%)

T5: An expansive view of a metal chain-link fence with a torn section creating a small hole. The fence stretches across the scene, held by firm metal posts, and the damage appears intentional yet not corroded. The composition emphasizes the open space around the fence.
CLIP: A panoramic view of a damaged chain-link fence in an urban area.

ID 32	**T** 3200	**A** 117 (3.66%)	**B** 0 (0.00%)

T5: An expansive view of a metal chain-link fence with a torn section creating a small hole. The background is filled with scattered construction materials, equipment, and debris. The fence stretches across the scene, held by firm metal posts, while the damaged area contrasts with the cluttered industrial setting behind it.
CLIP: A damaged chain-link fence in front of a busy construction site with scattered materials.

Comment: Some holes, too close POV

ID 33	**T** 3200	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: An expansive view of a metal chain-link fence with a torn section creating a small hole. The background consists of dense urban elements, including graffiti-covered walls, parked cars, and a mix of residential and commercial buildings. The fence stretches across the scene, partially blending with the chaotic cityscape behind it.
CLIP: A damaged chain-link fence in a crowded urban street filled with cars and buildings.

ID 34	**T** 3200	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: An expansive view of a metal chain-link fence with a torn section creating a small hole. Behind it, an overgrown junkyard is visible, filled with abandoned vehicles, rusted machinery, and scattered tires. The fence stretches across the chaotic background, its damage adding to the sense of neglect.
CLIP: A damaged chain-link fence in front of an overgrown junkyard with abandoned cars and machinery.

ID 35	**T** 64	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: An expansive view of a metal chain-link fence with a torn section creating a small hole. The fence stretches across the scene, held by firm metal posts, and the damage appears intentional yet not corroded. The background is an industrial laboratory with machines and desks.
CLIP: A panoramic view of a damaged chain-link fence in an industrial factory.

Comment: Nice backgrounds but too close POVs

ID 36	**T** 32	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A sweeping view of a long chain-link fence extending across an industrial environment, with a torn section creating many irregular and jagged hole. The damage appears deliberate, with bent and uneven wires forming sharp edges. The sturdy metal posts hold up the fence, contrasting with the fragile, damaged section that serves as the focal point
CLIP: A wide view of a damaged chain-link fence in an industrial environment.

Comment: Nice POVs but the background is too simple and virtually no background. Maybe I should increment the step size?

ID 37	**T** 32	**A** 1 (3.13%)	**B** 0 (0.00%)

T5: A sweeping view of a long chain-link fence extending across an industrial environment, with a torn section creating many irregular and jagged hole. The damage appears deliberate, with bent and uneven wires forming sharp edges. The sturdy metal posts hold up the fence, contrasting with the fragile, damaged section that serves as the focal point
CLIP: A wide view of a damaged chain-link fence in an industrial environment.

Comment: Migliorato ma lo sfondo è troppo semplice, e troppe immagini sono di background

ID 40	**T** 32	**A** 5 (15.63%)	**B** 1 (3.13%)

T5: A chain-link fence with visible damage, featuring a torn section forming an irregular hole. The scene is set in a cluttered industrial area with heavy machinery, metal shelves, workbenches, and scattered tools. The background includes large industrial buildings, storage containers, and dim artificial lighting casting shadows on the worn concrete floor
CLIP: A damaged chain-link fence with a large hole in an industrial area filled with machinery and scattered equipment

Comment: Steps size increased to 75. Pretty pictures; complex background, holes are way too big

ID 41	**T** 32	**A** 2 (6.25%)	**B** 0 (0.00%)

T5: A chain-link fence with a noticeable tear, forming an opening in the middle. The fence is situated within a vast industrial complex filled with workstations, tool racks, conveyor belts, and mechanical components. Overhead pipes run across the ceiling, while flickering industrial lights cast a cold glow over the scene. The atmosphere is dusty, with signs of wear and disrepair visible throughout the facility
CLIP: A broken chain-link fence with a large gap, located inside an industrial facility with workstations, scattered tools, and overhead pipes

Comment: Steps size increased to 75. Similar to 040

ID 42	**T** 32	**A** 3 (9.38%)	**B** 1 (3.13%)

T5: A chain-link fence with an irregularly cut hole, standing between the viewer and a busy industrial workspace. Beyond the fence, the area is packed with desks covered in blueprints and tools, metal shelves filled with spare parts, and machinery in various states of assembly. The walls are lined with old, rusted lockers, and dim fluorescent lights create a harsh contrast between illuminated areas and deep shadows
CLIP: A vandalized chain-link fence with a hole, enclosing an industrial workspace cluttered with desks, machinery, and storage units

Comment: Steps size increased to 75. Amazing pictures but zero holes, the fence is just interrupted

ID 43	**T** 32	**A** 0 (0.00%)	**B** 1 (3.13%)

T5: A weathered chain-link fence with visible signs of tampering, including a few small but deliberate cuts in the metal mesh. Behind it, an industrial complex with shipping containers, forklifts, and workbenches, illuminated by cold artificial lights. Graffiti and worn-out warning signs are posted along the fence, adding to the sense of abandonment
CLIP: A rusty chain-link fence with small, irregular holes, revealing a cluttered industrial yard filled with stacked crates, scattered tools, and dimly lit warehouses in the background

Comment: Steps size increased to 75. Nice backgrounds

ID 44	**T** 32	**A** 1 (3.13%)	**B** 3 (9.38%)

T5: A battered chain-link fence with several small openings, as if someone had attempted to break through multiple times. Beyond it, a narrow alleyway packed with debris, old machinery, and exposed pipes. The scene is dimly lit by neon reflections from nearby shop signs, contrasting with the dull, corroded metal of the fence
CLIP: A damaged metal fence with torn sections, standing in front of an urban alley filled with dumpsters, graffiti-covered walls, and flickering neon signs

Comment: Steps size increased to 75. No broken fences, unique cyberpunk atmosphere.

ID 45	**T** 32	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A construction area enclosed by a tall chain-link fence, partially cut in certain spots, allowing a glimpse of the chaotic worksite beyond. Stacks of wooden planks, piles of bricks, and idle excavators create a dense background. A "No Trespassing" sign hangs loosely, barely holding onto the rusted wire mesh.
CLIP: A fenced-off construction site with a chain-link barrier showing signs of forced entry, surrounded by heavy equipment and scattered debris

Comment: Nice complex backgrounds but no holes

ID 46	**T** 64	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A chain-link fence with large sections cut away, the metal edges bent outward as if someone had forcefully pushed through. Some parts of the fence are held together with haphazardly wrapped wires, while others dangle loosely. Behind it, an industrial facility with towering pipes, stacked wooden pallets, and old workbenches, illuminated by harsh artificial lighting.
CLIP: A heavily vandalized chain-link fence with multiple jagged holes and torn sections, barely standing in front of a cluttered industrial yard filled with rusted machinery, scattered tools, and flickering warning lights

ID 47	**T** 64	**A** 0 (0.00%)	**B** 0 (0.00%)

T5: A chain-link fence that has been repeatedly cut and patched, but remains visibly compromised. Several sections are missing, with sharp metal strands curling inward. Near the base, the fence has been bent and partially ripped from the ground. Beyond it, a dimly lit alleyway filled with debris, exposed pipes, and neon-lit storefronts casting a chaotic blend of colors onto the worn pavement
CLIP: A breached metal fence with gaping holes and twisted wires, standing in front of a dark alleyway cluttered with graffiti-covered dumpsters, broken glass, and old construction materials

Comment: Steps size increased to 75.

ID 48	**T** 3200	**A** 134 (4.19%)	**B** 75 (2.34%)

T5: A deteriorated chain-link fence surrounding a construction area, its metal wires severely bent and cut, leaving gaps large enough for someone to slip through. One section leans at an awkward angle, held up only by a single rusted post. Beyond the damaged barrier, a chaotic worksite with piles of bricks, broken scaffolding, and machinery covered in dust, all lit by dim floodlights casting long shadows
CLIP: A security fence torn open in multiple places, barely holding together in front of an abandoned construction site filled with overturned barriers, heavy equipment, and scattered debris

Comment: Steps size increased to 75. Nice POVs, nice holes, all night ambience. Great backgrounds, holes are too big.

ID 49	**T** 64	**A** 1 (1.56%)	**B** 0 (0.00%)

T5: A worn-down chain-link fence surrounding an active industrial yard, showing clear signs of tampering. Several small but deliberate cuts in the metal mesh create narrow gaps, just wide enough for a person to squeeze through. Beyond the damaged barrier, the site is cluttered with heavy machinery, workbenches covered in tools, and towering stacks of shipping containers, all under the bright midday sun
CLIP: A chain-link fence with multiple small breaches, its metal wires bent and severed, enclosing a busy industrial site with machinery, stacked containers, and scattered tools

Comment: Steps size increased to 75. Easy background and no holes

ID 50	**T** 64	**A** 4 (6.25%)	**B** 0 (0.00%)

T5: A chain-link fence in disrepair, enclosing a sprawling urban scrapyard. The mesh is punctured by multiple small holes, some appearing hastily cut, others showing signs of rust-induced breakage. Though not immediately obvious, the breaches are large enough for someone determined to squeeze through. Inside, the yard is a chaotic mix of abandoned cars, metal scraps, and wooden pallets, with sunlight casting sharp shadows on the rough concrete ground
CLIP: A vandalized fence with small openings, partially collapsed in places, enclosing a dense urban scrapyard filled with rusted vehicles, piles of metal, and scattered wooden pallets

ID 51	**T** 64	**A** 5 (7.81%)	**B** 0 (0.00%)

T5: A worn but still-standing chain-link fence marks the perimeter of a storage facility, showing clear evidence of tampering. The metal mesh is frayed and severed in multiple spots, with narrow openings just wide enough for an intruder to slip past. Beyond the fence, a bustling industrial yard is visible, filled with neatly stacked crates, rusted barrels, and forklifts maneuvering between storage units. The harsh daylight highlights the texture of the rough pavement and the dust settling in the air
CLIP: A damaged industrial fence with narrow cutouts, standing in front of a storage facility packed with large crates, barrels, and forklifts in motion

Comment: Steps size increased to 75. Simple background, nice holes.

Bibliography

[1] S.F. Capital. *Knightscope: Slow Rise of the Robots (rating upgrade)*. 2017 (cit. on p. 2).

[2] Robert Soler, Alae Moudni, Gabriel Roskowski, Xinrui Yu, Mikhail Gormov, and Jafar Saniie. «Autonomous Patrol and Threat Detection Through Integrated Mapping and Computer Vision». In: *2024 IEEE International Conference on Electro Information Technology (eIT)*. 2024, pp. 398–403. DOI: 10.1109/eIT60633.2024.10609884 (cit. on p. 2).

[3] N. Hemavathy, K. Arun, R. Karthick, A. P. Srikanth, and S. Venkatesh. «Night Vision Patrolling Robot with Sound Sensor using Computer Vision Technology». In: *International Research Journal of Engineering and Technology (IRJET)* 7 (2020) (cit. on p. 2).

[4] *Unitree Go2*. URL: https://www.unitree.com/go2 (cit. on p. 7).

[5] *Unitree Go2 Brochure*. URL: https://static.generation-robots.com/media/brochure-unitree-go2-en.pdf (cit. on pp. 7, 8).

[6] *NVIDIA Jetson for Next-Gen Robotics*. The title refers generically to the family AGX. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/ (cit. on p. 11).

[7] *NVIDIA DeepStream SDK*. URL: https://developer.nvidia.com/deepstream-sdk (cit. on p. 11).

[8] *NVIDIA JetPack SDK*. URL: https://developer.nvidia.com/embedded/jetpack (cit. on p. 11).

[9] *Install Jetson Software with SDK Manager*. URL: https://docs.nvidia.com/sdk-manager/install-with-sdkm-jetson/index.html (cit. on p. 11).

[10] Dr. Divyakant Meva John Anand. «A Comparative Study of Various Object Detection Algorithms and Performance Analysis». In: *International Journal of Computer Sciences and Engineering* 8 (Oct. 2020). DOI: 10.26438/ijcse/v8i10.158163 (cit. on p. 15).

[11] Kim Youngseop Lee Yong-Hwan. «Comparison of CNN and YOLO for Object Detection». In: *Journal of the Semiconductor Display Technology* (2020) (cit. on p. 15).

[12] *Ultralytics' YOLOv11*. URL: https://docs.ultralytics.com/it/models/yolo11/ (cit. on p. 15).

[13] Jürgen Beyerer Nils Friederich Andreas Specker. «Security Fence Inspection at Airports Using Object Detection». In: *Winter Conference of Applied Computer Vision 2024 (WACV '24)* (2023) (cit. on p. 15).

[14] *black-forest-labs/FLUX.1-schnell*. URL: https://huggingface.co/black-forest-labs/FLUX.1-schnell (cit. on p. 18).

[15] *stable-diffusion/stable-diffusion-v1-5*. URL: https://huggingface.co/stable-diffusion-v1-5/stable-diffusion-v1-5 (cit. on p. 18).

[16] *Announcing Black Forest Labs*. URL: https://blackforestlabs.ai/announcing-black-forest-labs/ (cit. on p. 18).

[17] *enfff/sorter*. Software developed leveraging GTK4 and Libadwaita. Tested on Arch Linux x86_64, Gnome 47.4, WM: Mutter (Wayland). URL: https://github.com/enfff/sorter (cit. on p. 21).

[18] *Label Studio*. An Open Source Data Labeling Platform. URL: https://labelstud.io/ (cit. on p. 21).

[19] *GStreamer Negotiation capabilities*. URL: https://gstreamer.freedesktop.org/documentation/additional/design/negotiation.html (cit. on p. 24).

[20] *GStreamer Negotiation capabilities*. DeepStream allows the use of configuration files for generating dynamically pipelines. These configuration files have a rigid structure and accept the settings specified in this documentation. URL: https://docs.nvidia.com/metropolis/deepstream/dev-guide/text/DS_ref_app_deepstream.html#configuration-groups (cit. on p. 25).