Vietnamese-German University

COPYRIGHT WARNING

This paper is protected by copyright. You are advised to print or download **ONE COPY** of this paper for your own private reference, study and research purposes. You are prohibited having acts infringing upon copyright as stipulated in Laws and Regulations of Intellectual Property, including, but not limited to, appropriating, impersonating, publishing, distributing, modifying, altering, mutilating, distorting, reproducing, duplicating, displaying, communicating, disseminating, making derivative work, commercializing and converting to other forms the paper and/or any part of the paper. The acts could be done in actual life and/or via communication networks and by digital means without permission of copyright holders.

The users shall acknowledge and strictly respect to the copyright. The recitation must be reasonable and properly. If the users do not agree to all of these terms, do not use this paper. The users shall be responsible for legal issues if they make any copyright infringements. Failure to comply with this warning may expose you to:

- Disciplinary action by the Vietnamese-German University.
- Legal action for copyright infringement.
- Heavy legal penalties and consequences shall be applied by the competent authorities.

The Vietnamese-German University and the authors reserve all their intellectual property rights.

VIETNAMESE – GERMAN UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

Frankfurt University of Applied Sciences

Faculty 2: Computer Science and Engineering

RESEARCH AND IMPLEMENTAION OF ARTIFICIAL INTELLIGENCE INTO THE INTERNET OF THINGS USING NVIDIA JETSON NANO

Full name: Bui Nhien Loc

Matriculation number: 15635

First supervisor: Dr. Tran Hong Ngoc

Second supervisor: Ms. Pham Ngoc Giau

BACHELOR THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF BACHELOR ENGINERRING
IN STUDY PROGRAM COMPUTER SCIENCE, VIETNAMESE – GERMAN UNIVERSITY, 2023

Binh Duong, Vietnam

Abstract

This bachelor's thesis explores the integration of Artificial Intelligence (AI) into Internet of Things (IoT) devices using NVIDIA Jetson Nano as the dedicated AI platform/edge device. The objective is to enhance the capabilities of IoT devices and enable them to autonomously perform sophisticated data analysis and decision-making tasks, improving overall efficiency and effectiveness.

The study begins with the inspiration and motivation for this project, followed by a literature review, assessing AI's potential applications, challenges, and benefits in IoT systems. The research then delves into the technical aspects of NVIDIA Jetson Nano, investigating its compact size, high-performance computing capabilities, and software libraries for seamless AI model deployment on resource-constrained IoT devices.

A prototype IoT device integrated with NVIDIA Jetson Nano is developed, capturing video footage and then applying AI algorithms for real-time analysis to extract the information needed for IoT decision-making. The results and foundation of this prototype can be used to expand and implement it in any facet of life. The topic is still being discussed and improved even after this thesis.

Keywords: Artificial Intelligence, Internet of Things, AIoT, facial recognition, Jetson Nano

Abbreviation

Acronyms	*Meaning*
AI	Artificial Intelligence
IoT	Internet of Things
AIoT	Artificial Intelligence of Things
QR code	Quick Response code
YOLO	You Only Look Once
GPU	Graphical Processing Unit
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
PC	Personal Computer
ARM	Advanced RISC Machines
RISC	Reduced Instruction Set Computing
LPDDR	Low Power Double Data Rate
SoC	System on Chip
TOPS	Tera Operations Per Second
TFOPS	Tera Floating-Point Operations Per Second
GFLOPS	Giga Floating-Point Operations Per Second
mAP	Mean Average Precision
AUC	Area Under Curve
IoU	Intersection over Union
Pascal VOC	Pascal Visual Object Classes
MS COCO	Microsoft Common Objects in Context
fps	frame per second
SORT	Simple Online and Realtime Tracking
MOT	Multi-Object Tracking
MQTT	Message Queuing Telemetry Transport
UUID	Universal Unique Identifier
POV	Point Of View
ZeroMQ	Zero Message Queue
ImageZMQ	Image Zero Message Queue
IP address	Internet Protocol address
CDN	Content Delivery Network
AVC	Advanced Video Coding
HEVC	High-Efficiency Video Coding
RTMP	Real-Time Messaging Protocol
HLS	HTTP Live Streaming
RTSP	Real Time Streaming Protocol
OBS	Open Broadcaster Software
VLC	VideoLAN Client
LPR	License Plate Recognition
OCR	Optical Character Recognition

Table 1: Acronyms table

List of Figures

List of Tables

Listings

Table of Contents

1 Introduction

As we transition towards Society 5.0, an extraordinary convergence between cyberspace and physical space comes to light. In the preceding era of Society 4.0, known as the age of information, the Internet acted as a conduit for searching, retrieving, and analyzing data from the vast realms of cyberspace. However, in Society 5.0, the focus shifts toward seamlessly integrating Artificial Intelligence (AI) into cyberspace. AI now takes center stage, empowered to analyze immense volumes of big data and subsequently relay the outcomes to humans in physical space through diverse mediums.

1.1 Idea evolution

The project's inception took root during my internship period, where I had an offer to develop a solution for a company's assembly line. The proposed solution involved scanning QR codes labeled on assembled products to track their progress and ensure quality control. However, as I delved deeper into the intricacies of the process, I noticed room for improvement and innovation.

The initial idea revolved around using the Internet of Things (IoT) technology to automate the detection and handling of defective products. Integrating IoT devices into the assembly line would automatically identify faulty items through the scanned QR codes. These flawed products would then be sorted out for removal or potential reusability, streamlining the production process and minimizing wastage.

As the concept grew and evolved, I realized that further enhancements could be made to the system. The integration of Artificial Intelligence emerged as a game-changer, promising to take the footage-capturing process to the next level. By deploying even a simple AI-powered video-processing algorithm, the system could analyze the footage captured by cameras stationed along the assembly line. This AI-driven analysis would enable extracting detailed information from the images, such as detecting subtle defects, assessing product quality, and even predicting potential issues before they escalate.

The information extracted through AI analysis would then be logged and integrated into the existing IoT system. This enriched data repository would provide invaluable insights into the assembly line's performance, quality trends, and potential areas for optimization. The synergy between AI and IoT would enable the system to make more informed decisions, enhance overall productivity, and ensure a seamless flow of operations.

1.2 Proposal

The version of the system proposed in this thesis boasts remarkable versatility and adaptability, making it amenable to various industries and personalized applications. Originally designed to detect defective products in an assembly line setting, the underlying AIoT infrastructure can seamlessly undergo modifications to cater to a wide range of use cases.

In industrial settings, the system's capabilities extend beyond defect detection. By fine-tuning the AI algorithms and reconfiguring the IoT components, it can be repurposed to enhance quality control in diverse manufacturing processes. Beyond manufacturing, the proposed system finds applications in security and surveillance domains. With minor adjustments, it can transform into a comprehensive security system, proficient in tracking individuals and vehicles in a designated area.

With the latest extension, it can also be used as a security checkpoint or just simply check for attendance. As a security checkpoint, the system can be strategically deployed at access points, entrances, or critical locations to monitor and regulate the flow of people or vehicles. The integration of facial recognition algorithms enables the system to accurately identify authorized personnel, granting them seamless entry while efficiently detecting and flagging unauthorized individuals. By cross-referencing captured data with an established database, the system can rapidly identify potential threats or persons of interest, enhancing overall security in sensitive environments such as airports, government buildings, or high-security facilities. Many companies with various successes.

All these advancements are made possible by harnessing the computational prowess and versatility of the NVIDIA Jetson Nano as the foundational base for the system. The Jetson Nano's compact size, high-performance computing capabilities, and power efficiency are pivotal in enabling seamless integration of AI and IoT technologies, empowering the system to excel in various complex tasks. It is also a great alternative to using an AI-powered camera. Similar systems have already been deployed, opting for non-AI cameras due to financial constraints.

1.3 Thesis Structure

For this thesis, the first thing on the agenda is to explore the world of Embedded Computing Systems by NVIDIA, more specifically, their line of NVIDIA Jetson. This part includes an overview of the NVIDIA Jetson line, comparing the specifications, pros, cons, and prices of the Jetson Nano, Jetson Xavier, and Jetson TX.

The AI aspect of the project is covered in two sections. The first section discusses YOLO (You Only Look Once) and its evolution, from its historical background to the latest version (e.g., YOLOv8). The latter is emphasized for its superiority in real-time object detection. The second section delves into YOLO model accuracy and its tracking function, highlighting how the latest version improves tracking capabilities for object detection, while also going over which version is the most suitable for the chosen edge device.

IoT database integration is explored as an integral part of the project, elucidating its role in managing, storing, and retrieving data for seamless AI integration. This project is also concerned with telecommunication as the actual user of the product will also want to see the detection/-footage live from the comfort of their device of choice. This aspect will go over the streaming concept, video compression, and streaming videos through the use of NGINX as a server.

Facial recognition, achieved through the combination of Facenet and YOLOv8, is explained

in-depth, highlighting the advantages it brings to the proposed system, and how this can be implemented in a school/workplace setting.

Lastly, the proposed system's architecture is presented, encompassing all the discussed components. The thesis concludes with an outlook on the system's generational development, foreseeing future advancements and potential areas of improvement to revolutionize the field of AI in IoT.

1.4 Related works

In recent years, the integration of AI with IoT has garnered significant attention due to its potential to revolutionize various industries and enhance the functionality of everyday devices. This section provides an overview of the existing literature and related research in the field of AIoT systems, highlighting key trends, methodologies, and contributions.

1.4.1 AIoT system to monitor traffic

In June 2023, a deployment of four cameras, each equipped with an AI Box, was initiated to monitor the Thai Ha crossing bridge in Ha Noi. Despite the article's title suggesting the utilization of AI cameras, the system actually employs standard cameras integrated with IoT boxes housing AI modules for processing the camera feed. The primary objective of this setup is to identify vehicles, primarily trucks and buses, exceeding a height of 2.2 meters. Upon detection of a qualifying object, an LED display situated at the bridge's entrance activates, discouraging the vehicle from proceeding onto the bridge. Furthermore, the display promptly switches to showcase the detected vehicle's license plate number, directly addressing the driver. This system operates with the imperative of making swift predictions to ensure accurate warnings for incoming traffic. It is worth noting that this approach is driven by cost-effectiveness, as AI cameras are considered prohibitively expensive. Currently undergoing testing, the system has ambitions to extend its application to other areas in the future.

1.4.2 Security system using Facial Recognition

MyAloha, a dynamic tech company headquartered in the vibrant landscape of Vietnam, is dedicated to pushing the boundaries of technology and delivering innovative solutions that redefine the way we interact with our digital world. In a world where traditional security measures often prove inadequate for the demands of a dynamic environment, MyAloha introduces its cutting-edge FaceID technology. This revolutionary system brings forth a paradigm shift in security, enabling buildings and facilities to monitor entries and exits with unparalleled precision. Beyond mere surveillance, FaceID possesses the astute capability to discern authorized personnel, effectively transforming the binary code of 0s and 1s into the seamless operation of opening doors and gates, thereby streamlining access management. Furthermore, it serves as a vigilant guardian, swiftly identifying individuals of concern and promptly alerting the appropriate authorities.

The mechanics of this innovative system are as fascinating as they are effective. New faces

are seamlessly incorporated into the system by uploading a portrait of an individual, which subsequently serves as a foundational dataset. The magic then unfolds as the system employs this image to meticulously construct a highly detailed 3D model of the person's head. This sophisticated process transforms the abstract concept of facial recognition into a tangible, three-dimensional representation, enhancing the system's accuracy and reliability.

1.4.3 Check-in by Facial Recognition

Amidst the COVID-19 pandemic, Vinpearl, a renowned chain of hotels, resorts, and entertainment destinations in Vietnam, took an innovative step that set a pioneering precedent in the country's hospitality industry. In response to the pressing need for reduced physical contact and heightened safety measures, Vinpearl introduced a groundbreaking Contactless Check-In System utilizing Facial Recognition technology, a first of its kind in Vietnam.

The ingenious system was meticulously designed to minimize direct human interaction while concurrently enhancing the efficiency of the booking and reservation process. Since its initial implementation in 2019, the system has undergone a remarkable evolution, steadily replacing traditional staff-assisted procedures with automated ticket and reservation verification processes.

In 2023, Vinpearl's Contactless Check-In System reached an apex of sophistication. After arriving at the resort, the guests check in by having their picture taken and this data can be used for other services. Guests can now seamlessly book a wide array of services offered by Vinpearl, make advance payments, and subsequently, upon arrival, have their faces scanned by a facial recognition system. This innovative process not only ensures a swift and secure check-in but also cross-references the guest's facial data to ascertain the availability of the requested services.

2 NVIDIA Jetson

The section goes over the differences between the Jetson products that NVIDIA provided and how they affect the final decision of using the Jetson Nano line for the project.

2.1 What is the NVIDIA Jetson line?

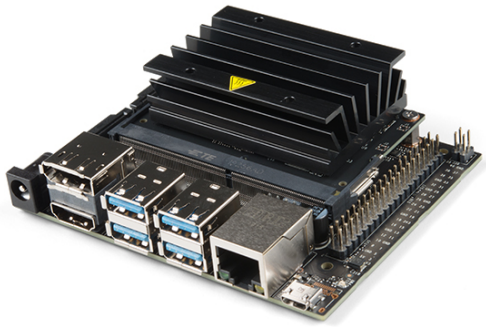
2.1.1 Overview

The NVIDIA Jetson line is a series of highly efficient, low-power compute modules embedded in computing platforms that have been specifically designed to elevate AI at the edge. This line has been developed by NVIDIA, a well-known to be GPU manufacturer. They are also a renowned pioneer in the fields of AI and GPU computing, with the sole aim of providing a comprehensive range of hardware and software solutions that cater to the diverse needs of AI-driven applications across a wide range of industries.

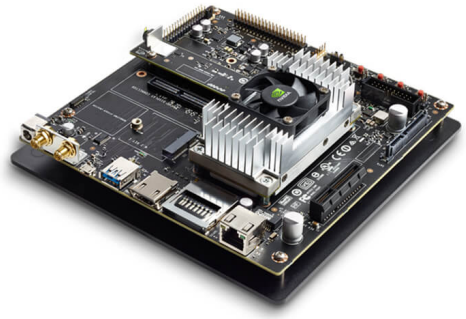
Figure 1: NVIDIA: World Leader in Artificial Intelligence Computing

In 2014, NVIDIA introduced the first Jetson development kit, the NVIDIA Jetson TK1. This platform was a game-changer for embedded AI computing. The Tegra K1, NVIDIA's first mobile processor with advanced features and architecture similar to a modern desktop GPU, paved the way for embedded devices to use the same CUDA code as desktop GPUs, resulting in similar levels of GPU-accelerated performance. This made it a popular choice among over 100,000 developers.

Since then, the Jetson line has seen several iterations, each boasting increased performance and enhanced features to cater to the growing AI needs of diverse applications. Currently, some iterations, from lowest to highest computing power, provided on the official NVIDIA website include Jetson Nano, Jetson TX2, Jetson Xavier, and the latest line, Jetson Orin.



(a) Jetson Nano



(b) Jetson TX2

(c) Jetson Xavier

(d) Jetson Orin

Figure 2: The Jetson line interations

2.1.2 Specifications and Statistics

a) Jetson Nano

The NVIDIA Jetson Nano is a remarkable embedded computing platform. With the tag line: "Bringing the Power of Modern AI to Millions of Devices.", it is perfect for beginners to start learning about AI and robotics in the real world. Introduced by NVIDIA in March 2019, the Jetson Nano has revolutionized the landscape of edge AI computing, empowering developers, researchers, and enthusiasts to deploy advanced artificial intelligence applications in a compact and cost-effective form factor.

Although it is said to have the lowest computing power among NVIDIA's current offerings, it still boasts a significant amount of specifications. At the heart of the Jetson Nano lies a quad-core ARM Cortex-A57 CPU, delivering substantial processing power to handle complex

tasks efficiently. Complementing this CPU is a 128-core NVIDIA Maxwell GPU, which provides accelerated computing for AI workloads and computer vision tasks, ensuring swift and accurate data analysis.

With two options for the memory of 2GB and 4GB of LPDDR4 RAM, the Jetson Nano can efficiently handle large datasets and memory-intensive AI models. Depending on the size and scope of the project, it is important to choose ample memory capacity that allows for seamless multitasking and facilitates real-time inference in AI applications.

The Jetson Nano and many of its peer carries a standout feature of support for popular AI frameworks like TensorFlow, PyTorch, and Caffe, enabling seamless integration into existing AI workflows. Just as alluded to in the previous section, it also offers NVIDIA's CUDA-X just like any modern computer that includes an NVIDIA GPU. This allows the machine to harness the power of the GPU to accelerate AI training and inference.

Despite its exceptional performance, the Jetson Nano remains remarkably energy-efficient, consuming as little as 5 watts under typical loads. This is essential and the deciding factor for using a system like this instead of a laptop or personal computer. It is expected for the system to work around the clock, with hardly any power consumption to prolong operation without compromising on performance.

b) Jetson TX2

Also part of the NVIDIA Jetson line is the Jetson TX2 series. As the successor of the discontinued Jetsono TX1, it is also an excellent option for AI and embedded IoT applications. Introduced back in March 2017, the Jetson TX2 series represents a significant advancement in the Jetson family, catering to the demands of high-performance AI applications, computer vision tasks, and autonomous machines.

Significant improvements compared to the Jetson Nano are the TX2's 256-core NVIDIA Pascal GPU, combined with a dual-core NVIDIA Denver 2 CPU and a quad-core ARM Cortex-A57 CPU. This particular multi-core architecture is designed to effectively handle tasks that require high performance as well as low power, allowing for smooth multitasking and real-time data processing.

As it is called the Jetson TX2 series, there are multiple versions of the TX2, including TX2 NX, TX2 4GB, TX2, and TX2i. Similar to the Jetson Nano are the multiple options for LPDDR4 RAM: 4GB, 8GB, and 8GB with ECC Support.

Having more computing power, however, will cause the system to consume more energy to keep up with its exceptional performance capabilities, around 7.5 watts under typical loads. This is still a major improvement compared to the typical energy consumption of a laptop or PC.

c) Jetson Xavier

The Xavier architecture is shared by two sub-series, which are the NVIDIA Jetson AGX Xavier Series and Jetson Xavier NX Series. Introduced at around the same time, during the winter of 2018, they represent two powerful and innovative lines of embedded computing platforms designed to accelerate AI at the edge.

Before the introduction of the Orin architecture, the AGX line was considered to be the epitome of AI performance and versatility that the Jetson was capable of providing. Powered by the NVIDIA Xavier SoC, it features an octa-core NVIDIA Carmel CPU, a 512-core NVIDIA Volta GPU, and LPDDR4x RAM.

Followed just behind the AGX is the NX line. Sharing the NVIDIA Xavier Soc, while having a 384-core NVIDIA Volta GPU, and a 6-core NVIDIA Carmel CPU. Not a major downgrade from AGX, considering the NX consumes around 7.5 watts on the low-end and around 10W on the high-end, while the AGX consumes about 10W all around.

With these features, combined with the mentioned benefits of the Jetson line, the Xavier line boasts higher AI performance and more memory, making it suitable for computationally intensive, while offering more energy-efficient and compact alternatives without compromising AI capabilities.

d) Jetson Orin

Similar to the Xavier line, many sub-series of the Orin line are being introduced during 2022 and 2023. Being the newest in the Jetson line, these machines carried a lot of computing power.

The products are still being released and developed for the near future, so the information regarding them is scarce and limited. That being said, with the released varieties, they boast the broadest range of selections for AI development compared to their predecessors.

On the low end of the spectrum, the Jetson Orin Nano 4GB provides a 512-core NVIDIA Ampere GPU, with a 6-core Arm Cortex CPU. Meanwhile, the Jetson AGX Orin Developer Kit, a high-end option, has a 2048-core NVIDIA Ampere GPU, and a 12-core Arm Cortex CPU.

All of them are equipped with the latest line of LPDDR5 RAM, with the lowest option providing 4GB, and the highest being 64GB. This combined the previous versions' benefits with extra upgrades and improvements, the Orin line is shaping up to be the future of AI and edge development.

2.2 How to evaluate the AI performance of a machine

Evaluating the performance of AI on a computer involves assessing its ability to handle complex computational tasks, such as machine learning, deep learning, and neural network processing. AI performance evaluation is crucial for determining a computer's suitability for running AI workloads, such as training models, executing inference, and real-time processing of data. To accurately measure AI performance, various metrics are considered, including speed, throughput, accuracy, and efficiency.

One key metric used to quantify AI performance in modern computers and AI accelerators is TOPS, which stands for Tera Operations Per Second. TOPS represents the number of trillion operations that a computer or AI accelerator can perform in a single second. It serves as a crucial performance indicator for AI tasks, where massive parallel processing capabilities are required for tasks like neural network computations.

In the context of AI, operations refer to calculations performed during the execution of algorithms, particularly in deep learning models. These operations typically involve matrix multiplications, additions, and other mathematical operations that are fundamental to neural network computations.

TOPS provides a standardized way to compare the processing power of different AI accelerators and GPUs, making it easier for researchers, developers, and hardware manufacturers to understand the computational capabilities of a device. It allows them to make informed decisions about which hardware is best suited for their specific AI workloads, taking into account factors like model complexity, data size, and real-time requirements.

In the next section, a comparison between the different Jetson machines will be drawn. Along with TOPS, there are a few other measurements to determine the AI performance, those being GFLOPS and TFLOPS. GFLOPS stands for Giga Floating-Point Operations Per Second. Similar to TOPS, it is a measure of computing performance, but at a smaller scale (1 TOPS is equivalent to 1000 FLOPS). Meanwhile, TFLOPS stands for Tera Floating-Point Operations Per Second. Like TOPS, it is a measure of computing performance, but it specifically focuses on floating-point operations, which are arithmetic operations involving real numbers with decimal points.

2.3 Comparison: the pros and cons

This section serves as a comprehensive comparison between the NVIDIA Jetson products, including their advantages and disadvantages. Having discussed around 20 unique products spanning 4 system architectures, only the notable module of each architecture will be taken into account. Also of note is the early reviews for other Jetson Orin lines, as they are still being released, the comparison made might not be accurate, retrospectively.

	Jetson Nano Developer Kit	**Jetson TX2 NX**	**Jetson Xavier NX 8GB**	**Jetson Orin Nano 4GB**
AI Performance	472 GFLOPS	1.33 TFLOPS	21 TOPS	20 TOPS
GPU	128-core NVIDIA Maxwell™ GPU	256-core NVIDIA Pascal™ GPU	384-core NVIDIA Volta™ GPU with 48 Tensor Cores	512-core NVIDIA Ampere™ GPU with 16 Tensor Cores
CPU	Quad-core ARM Cortex-A57 MPCore processor	Dual-Core NVIDIA Denver 2 64-Bit CPU Quad-Core ARM Cortex-A57 MPCore processor	6-core NVIDIA Carmel Arm v8.2 64-bit CPU 6MB L2 + 4MB L3	6-core Arm Cortex-A78AE v8.2 64-bit CPU 1.5MB L2 + 4MB L3
Memory	4 GB 64-bit LPDDR4 25.6 GB/s	4GB 128-bit LPDDR4 51.2GB/s	8GB 128-bit LPDDR4x 59.7GB/s	4GB 64-bit LPDDR5 34 GB/s
Storage	16GB eMMC 5.1	16GB eMMC 5.1	16GB eMMC 5.1	(Supports external NVMe)
Video encoding	1x 4K30 (H.265) 2x 1080p60 (H.265)	1x 4K60 (H.265) 3x 4K30 (H.265) 4x 1080p60 (H.265)	2x 4K60 (H.265) 4x 4K30 (H.265) 10x 1080p60 (H.265) 22x 1080p30 (H.265)	1080p30 supported by 1-2 CPU cores
Video decoding	1x 4K60 (H.265) 4x 1080p60 (H.265)	2x 4K60 (H.265) 7x 1080p60 (H.265) 14x 1080p30 (H.265)	2x 8K30 (H.265) 6x 4K60 (H.265) 12x 4K30 (H.265) 22x 1080p60 (H.265) 44x 1080p30 (H.265)	1x 4K60 (H.265) 2x 4K30 (H.265) 5x 1080p60 (H.265) 11x 1080p30 (H.265)

Power Consumption	5-10W	7.5W - 15W	10W - 20W	7W - 10W
Pros	- Cheap, suitable for newbies, amateurs.	- Mature, robust platform that offers a lot of options.	- High GPU power and processing.	- A powerhouse that can handle simulations, the most modern AI module, and 4K video playback.
Cons	- Doesn't have built-in WIFI. - Acceptable to power failure and will turn off when running heavy applications.	- Hard to set up.	- Quite pricy. - Higher power consumption and heat generation compared to other products.	- Not beginner friendly - Very pricy while being very overqualified for everyday usage.

Table 2: NVIDIA Jetson line comprehensive comparison

2.4 Why choose the Jetson Nano?

Based on the title and context of this thesis, it is clear that the Jetson Nano was selected for this project. The decision was made based on two critical factors. Firstly, the cost of the machines was taken into account, as this is also an alternative to expensive AI cameras due to financial constraints. Secondly, even though the Jetson Nano is the least powerful system among the options, it is sufficient for the project's foundation. If it can handle the inference time and computing power without any issues, more powerful machines can also handle the task.

3 You Only Look Once

You Only Look Once (YOLO) is one of the most popular model architectures and object detection algorithms. This will act as the AI part of the AIoT system.

3.1 What is YOLO?

3.1.1 Overview

Object detection is a fundamental task in computer vision, enabling machines to identify and locate objects within images or video frames. Over the years, various techniques and algorithms have been developed to tackle this challenging problem. One groundbreaking approach that revolutionized the field of object detection is YOLO.

The Yolo algorithm was introduced back in 2016. Before that, object detection algorithms typically employed multi-stage pipelines, which involved region proposal methods to identify potential object regions followed by classification and refinement. While these methods achieved good accuracy, they were computationally expensive and time-consuming, limiting their real-time applicability.

In contrast, YOLO approached object detection as a single regression problem, using a deep neural network to directly predict bounding boxes and class probabilities for objects within an image. By considering the entire image in one shot, YOLO achieved a significant speed boost, allowing it to process images at a near real-time speed.

Since then, there have been many versions of YOLO, with the latest being YOLOv8. The next section will discuss each version and its advancements throughout the generations.

3.1.2 YOLO throughout the years

a) YOLOv1

The YOLO network, specifically YOLOv1, was the first object detection model to integrate bounding box drawing and class label identification into a single, differentiable network.

Deep learning-based detection methods can be divided into two categories: two-stage detection algorithms, like RCNN and Fast-RCNN, which make multiple-stage predictions, and one-stage detectors like SSD, EfficientDet, and YOLO. YOLO is not the only one-stage detection model, but it is generally more efficient than others in terms of speed and accuracy. If we view the detection problem as a one-step regression approach for determining the bounding box, YOLO models are typically faster and smaller, making them easier to learn and deploy, particularly on devices with limited computing resources.

Despite its speed advantage, YOLOv1 had some limitations in accurately detecting small objects and handling overlapping objects. These drawbacks led to missed detections and imprecise localization, motivating the development of subsequent versions of YOLO.

b) YOLOv2

In 2017, YOLOv2 was introduced as a significant upgrade to YOLOv1. YOLOv2 improved upon YOLOv1 by introducing innovative advancements to address its weaknesses.

Several iterative enhancements were made to the architecture of YOLOv2, such as BatchNorm, higher resolution, and anchor boxes. The addition of anchor boxes enabled the algorithm to accurately predict objects of different sizes and aspect ratios.

c) YOLOv3

With the release of YOLOv3 in 2018, the YOLO (You Only Look Once) series made significant strides in advancing object detection capabilities. One of the key advancements in YOLOv3 was the addition of objectivity estimation to bounding box predictions. This improvement aimed to enhance the quality of object localization. By introducing objectivity estimation, the model became better at determining the presence of objects within bounding boxes, reducing false positives, and improving the accuracy of object detection.

The addition of skip connections to Darknet-53 improved YOLOv3's performance by allowing information to flow more easily between layers. This enabled the model to detect objects of varying sizes with greater accuracy at three different levels of detail. This multi-scale approach overcame limitations in previous versions and improved the localization of small objects.

d) YOLOv4 and YOLOv5

In 2020, YOLOv4 was launched after extensive experimentation and research. It combines several new techniques to enhance the accuracy and speed of the convolutional neural network. Extensive experiments were conducted in the paper introducing this version of YOLO and testing various GPU architectures. The results showed that YOLOv4 outperforms all other object detection network architectures in terms of both speed and accuracy.

In the same year, YOLOv5 was released with even more improvements, making it one of the official state-of-the-art models. YOLOv5 provides a range of object detection architectures that come pre-trained on the MS COCO dataset. This model is also natively implemented in PyTorch, which removes the limitations of the Darknet framework that was previously based on the C programming language and not built in terms of production environments. This allows for greater ease of use and support for production environments, making it a popular choice among developers.

e) YOLOv6 and YOLOv7

In 2022, YOLOv6 and YOLOv7 were released. With YOLOv6, it is an improved version of the previously popular YOLO trunk and neck. It has been designed to cater to hardware constraints and has introduced the EfficientRep Backbone and Rep-PAN Neck. Notably, YOLOv6 has sepa-

rated the classification and box-regression heads, which has demonstrated enhanced performance compared to previous versions.

Meanwhile, YOLOv7 is one of the cutting-edge object detectors in the YOLO family. This model contains all the most advanced deep neural network training techniques. YOLOv7 builds upon the advancements made in object detection technology through research on memory storage and gradient propagation. It considered these factors in its development, with a focus on using the E-ELAN last layer aggregation, an extended version of the ELAN compute unit. These developments hold promise for further enhancing the capabilities of object detection technology in various industries.

f) YOLOv8

Finally, YOLOv8, launch in early 2023. It is a cutting-edge model that takes the best of prior YOLO versions and adds new advancements to make it even better. The enhanced performance and versatility are sure to make a big difference for users.

Spearheaded by Ultralytics, a major advantage for version 8 is the framework supports all previous YOLO models, making it easy for users to switch between different versions and evaluate their performance. YOLOv8 is still being continuously developed and improved, but early results have shown this to be a major advancement for the YOLO family.

3.2 How to evaluate the accuracy of object detection model?

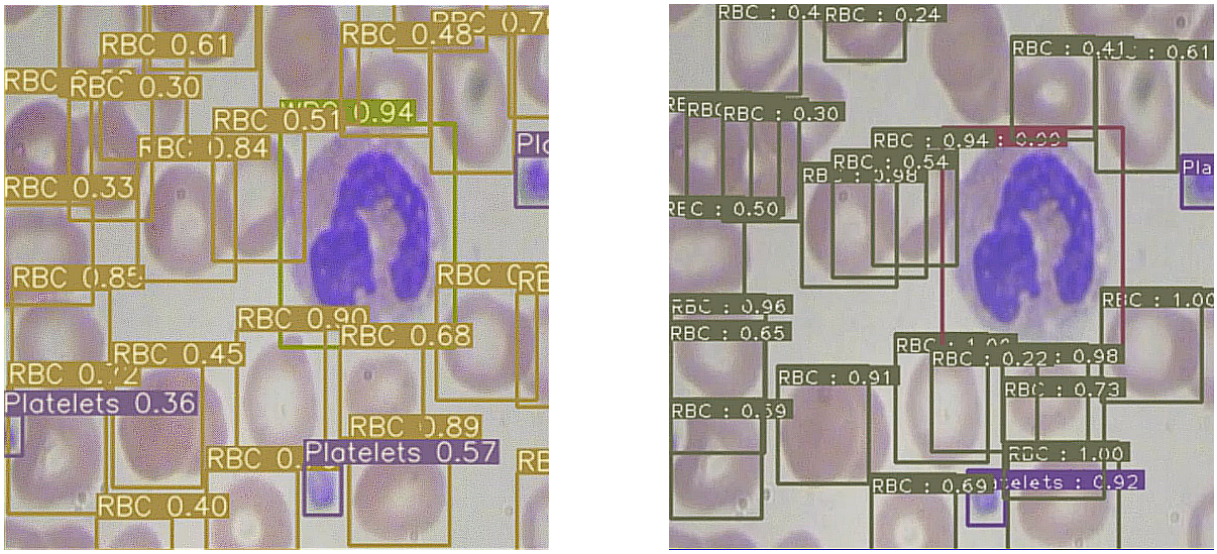
To evaluate an object detection model like YOLO, performance metrics like YOLO can be used to see how they can be compared with each other. This category alone can have multiple methods for the effectiveness of each model. The method of notice for this application is the Mean Average Precision (mAP).

Mean Average Precision is a widely used performance metric for evaluating the accuracy of object detection and instance segmentation models in computer vision. It is a comprehensive and robust evaluation metric that considers both precision and recall across multiple confidence thresholds. mAP is particularly useful for assessing the performance of object detection models, where precise localization and accurate classification of objects are essential. It is a great way to see how the model performs compared to other models on the same test dataset.

3.2.1 Comparing between two models

To understand it better, here is an example of two detectors working on the same test datasets. Figure 3 represents the red blood cells and platelets detection models powered by YOLOv3 and EfficientDet-D0, respectively. To determine the accuracy of these two models is quite simple for an average person, as the YOLOv3 identifies the cells more accurately, while EfficientDet-D0 wrongly detects way more cells, especially the cells close to each other, and not being able to

detect the ones only partially visible in the frame. However, it is essential to convert them into numerical values throughout the dataset for scientific analysis and comparison.



(a) YOLOv3

(b) EfficientDet-D0

Figure 3: Comparison between 2 object detection models on the same dataset

3.2.2 Precision-Recall Curve

This led to another definition: The Precision-Recall Curve. Precision-Recall Curve is a way to visualize how the module is performing as the confidence threshold is decreased. Going back to figure 3, next to each detection is a floating number representing the confidence of the model for that prediction. It is then we can tell if the confidence level is high (closer to 1), then the module is pretty certain about that prediction. Of course, predictions only values from 0.1 and 0.2 should be weeded out.

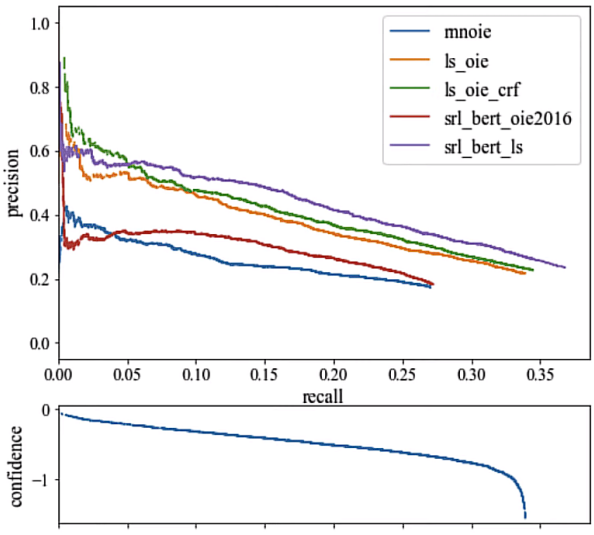


Figure 4: Example of precision and confidence changing according to confidence threshold

This is where the confidence threshold is established. Looking at figure 5, there are 2 new metrics: recall and precision. The recall is a measure of all the true positives, and how many predictions the module correctly made. While precision is the ratio of correct predictions. In other words, out of all the detections, how many of them are correct?

This is why when the confidence threshold is lower, we trade having more predictions, but a lot of them will be incorrect. This is why the graph is at a downward slop, as the confidence is lower, there are more predictions, which then lower the precision, and raise the recall.

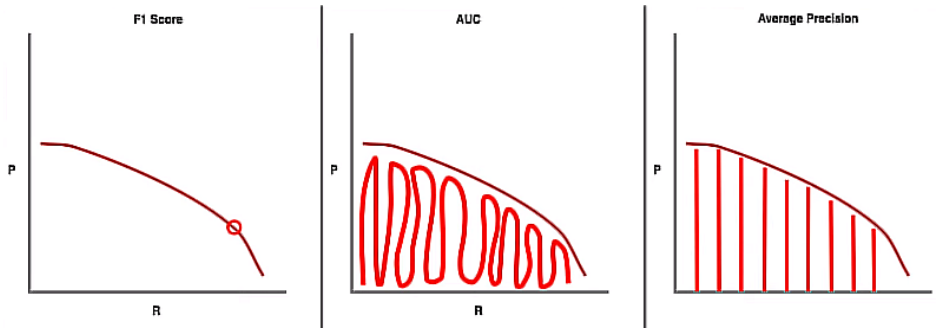


Figure 5: Example of value gathered from Precision-Recall Curve

From the Precision-Recall Curve, 3 main values can be extracted. The first value is the F1 Score, which is a single estimate of the Precision-Recall Curve where the Precision and Recall are multiplied to a single point. Next is the Area Under Curve, which is the total area under the Precision-Recall Curve. Finally, and the most important is the Average Precision metric. It looks at the proficient floating point estimate at various points along the curve, and the results will be used to calculate the Mean Average Precision.

3.2.3 Intersection over Union

To test and evaluate a module, we run detections on a test dataset. This dataset has already been annotated with the correct object and the module's job is to make detections to be compared with these established boxes. The IoU (Intersection over Union) is a metric to set the leniency to determine if the module made the correct prediction.

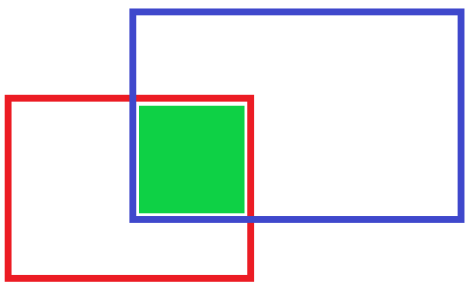


Figure 6: IoU illustrated

This value represents the value between the ground truth bounding box of the test dataset and the bounding box drawn by the module. In figure 6, it is illustrated as the green box in the middle. By increasing the IoU, we force the module to make more accurate predictions, while decreasing will generate the opposite effect. From there, an mAP curve can be drawn from the multiple values of IoU.

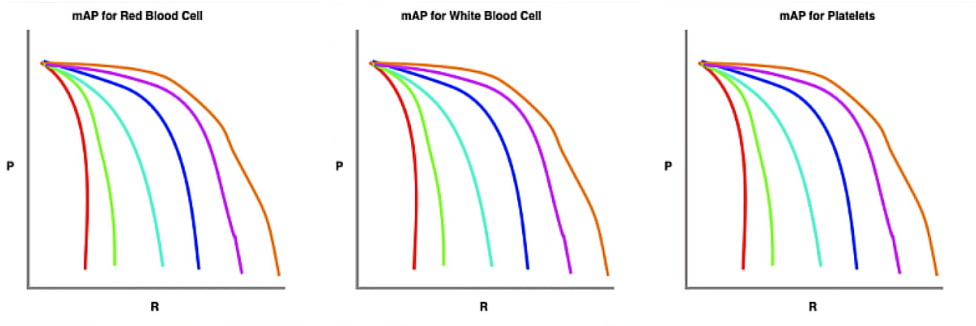


Figure 7: Example of mAP curves

The example of mAP curves from graph 7 can be simplified by seeing the orange line as the run with a low IoU value (from 0.05 to 0.1) so the curve is higher. However, the situation differs for the red line, which exhibits an IoU value ranging from 0.8 to 0.9. This stricter threshold leads to a more rapid decline in the curve.

3.2.4 Calculating mAP

From the result of the mAP curves, the mAP is calculated across various slew of IoU thresholds and multiple different classes. Doing so will give a robust way to look at the dataset. This is also a way to negate the problem of different models performing better at different IoU values or having an edge over detecting certain objects.

3.3 Comparison of different YOLO models

Using the mAP and fps metrics, the models are compared to determine the version for this project. Note that the values are calculated on two main datasets, Pascal VOC and MS COCO, and calculated over various devices. The results may vary depending on the changes in these variables.

Version	YOLOv1	YOLOv2	YOLOv3	YOLOv4	YOLOv5	YOLOv6	YOLOv7	YOLOv8
mAP	36.4	78.6	57.9	65.7	68.9	49.1	54.8	60
fps	45	67	46	62	125	267	83	500

Table 3: YOLO family comprehensive comparison

From the table, a few interesting aspects appear. YOLOv1 and YOLOv2 were trained and tested on the Pascal VOC dataset, which can make YOLOv1 and YOLOv2 seem more proficient than they were. Meanwhile, YOLOv6 trades off having a lower inference time to have a worse mAP compared to its predecessor. In the end, 2 models stand out for the application, YOLOv7 and YOLOv8. In the next section, these two modules will be implemented in the Jetson Nano to compare their performances.

4 YOLO on the Jetson Nano

The subsequent phase involves the critical evaluation of the determined YOLO models, namely YOLOv7 and YOLOv8, in terms of their performance when deployed on the Jetson Nano platform.

4.1 Methodology

The models were trained and evaluated using the MS COCO dataset. Then on the Jetson Nano, the necessary libraries are installed, and to keep the test as simple as possible, additional speeding-up methods are not included. The models are compared using weights of similar capacity, with the range of inference time being charted down to calculate the fps, and the mAP is noted in a separate validation. Also to compare the performance of the Jetson Nano, a machine with an AMD Ryzen 7 6500H with Radeon Graphics (16 vCore) CPU machine, and a separate run accelerated by an NVIDIA GeForce RTX 3050 GPU. Additional configurations are also added to set its effects on mAP and fps.

4.2 Results

After an extensive testing phase, here are the results.

4.2.1 Laptop CPU test

Configurations				fps			Inference Time (ms)				mAP
YOLO	images size (px)	confidence	IoU	Min	Mean	Max	Range	Min	Mean	Max	mAP
YOLOv7-tiny	320	0.5	0.45	82	71.7	63.7	12.2-15.7	12.2	13.95	15.7	0.352
	320	0.25	0.75	71.4	67.6	64.1	14-15.6	14	14.8	15.6	0.356
	320	0.5	0.75	80	71.2	64.1	12.5-15.6	12.5	14.05	15.6	0.354
	320	0.25	0.45	73	68.3	64.1	13.7-15.6	13.7	14.65	15.6	0.355
	480	0.5	0.45	45.2	42.1	39.4	22.1-25.4	22.1	23.75	25.4	0.364
	480	0.25	0.75	51.3	46.7	42.9	19.5-23.3	19.5	21.4	23.3	0.366
	480	0.5	0.75	55.2	48.3	42.9	18.1-23.3	18.1	20.7	23.3	0.364
	480	0.25	0.45	50	44.6	40.3	20-24.8	20	22.4	24.8	0.365
	640	0.5	0.45	32.2	30.1	28.2	31.1-35.4	31.1	33.25	35.4	0.368
	640	0.25	0.75	31.1	28.7	26.7	32.2-37.4	32.2	34.8	37.4	0.367
	640	0.5	0.75	32.2	29.2	26.8	31.1-37.3	31.1	34.2	37.3	0.370
	640	0.25	0.45	32.4	29.3	26.8	30.9-37.3	30.9	34.1	37.3	0.368
YOLOv7	320	0.5	0.45	16.5	15.9	15.3	60.5-65.4	60.5	62.95	65.4	0.485
	320	0.25	0.75	16.7	16	15.3	60-65.3	60	62.65	65.3	0.484
	320	0.5	0.75	16.6	15.6	14.7	60.2-67.9	60.2	64.05	67.9	0.487
	320	0.25	0.45	18	17	16.1	55.5-62.1	55.5	58.8	62.1	0.486
	480	0.5	0.45	9	8	7.1	110.7-140	110.7	125.35	140	0.496
	480	0.25	0.75	8.8	8.5	8.3	114-120	114	117	120	0.497
	480	0.5	0.75	8.9	8.7	8.4	112-119	112	115.5	119	0.499

	480	0.25	0.45	9.2	8.7	8.3	109-120	109	114.5	120	0.497
	640	0.5	0.45	4.9	4.7	4.6	206-217	206	211.5	217	0.511
	640	0.25	0.75	5	4.8	4.6	201-217	201	209	217	0.510
	640	0.5	0.75	5	4.8	4.7	202-213	202	207.5	213	0.512
	640	0.25	0.45	5.2	5.1	5	192-201	192	196.5	201	0.511
YOLOv7-e6	320	0.5	0.45	11.1	10.6	10.1	90-99	90	94.5	99	0.540
	320	0.25	0.75	10.2	9.8	9.5	98.4-105	98.4	101.7	105	0.538
	320	0.5	0.75	11	10.5	10	90.8-99.8	90.8	95.3	99.8	0.541
	320	0.25	0.45	10.5	10.1	9.6	94.8-103.9	94.8	99.35	103.9	0.540
	448	0.5	0.45	7.5	7.3	7	133-142	133	137.5	142	0.553
	448	0.25	0.75	7.6	7.3	6.9	131-144	131	137.5	144	0.552
	448	0.5	0.75	7.5	7.3	7.1	133-140	133	136.5	140	0.555
	448	0.25	0.45	7.4	7	6.6	135-151	135	143	151	0.554
	640	0.5	0.45	4.1	4.1	4	243-249	243	246	249	0.558
	640	0.25	0.75	4.2	4.1	4	240-250	240	245	250	0.557
	640	0.5	0.75	4	3.9	3.8	252-260	252	256	260	0.559
	640	0.25	0.45	4.1	4	4	246-250	246	248	250	0.559
YOLOv8n	320	0.5	0.45	70.9	67.1	63.7	14.1-15.7	14.1	14.9	15.7	0.486
	320	0.25	0.7	76.9	72.5	68.5	13-14.6	13	13.8	14.6	0.527
	320	0.5	0.7	80	72.7	66.7	12.5-15	12.5	13.75	15	0.486
	320	0.25	0.45	87.7	72.5	61.7	11.4-16.2	11.4	13.8	16.2	0.536
	480	0.5	0.45	66.2	57	50	15.1-20	15.1	17.55	20	0.555
	480	0.25	0.7	58.8	53.8	49.5	17-20.2	17	18.6	20.2	0.632
	480	0.5	0.7	58.1	51.8	46.7	17.2-21.4	17.2	19.3	21.4	0.555
	480	0.25	0.45	64.1	56.5	50.5	15.6-19.8	15.6	17.7	19.8	0.639
	640	0.5	0.45	49.5	42.6	37.3	20.2-26.8	20.2	23.5	26.8	0.564
	640	0.25	0.7	47.4	38.8	32.8	21.1-30.5	21.1	25.8	30.5	0.615
	640	0.5	0.7	47.6	41.9	37.5	21-26.7	21	23.85	26.7	0.560
	640	0.25	0.45	47.4	40	34.6	21.1-28.9	21.1	25	28.9	0.626
YOLOv8m	320	0.5	0.45	12	11.8	11.5	83.2-86.9	83.2	85.05	86.9	0.686
	320	0.25	0.7	12.9	12.7	12.6	77.8-79.6	77.8	78.7	79.6	0.727
	320	0.5	0.7	12.3	11.9	11.5	81.2-86.6	81.2	83.9	86.6	0.686
	320	0.25	0.45	12.7	12	11.5	78.7-87.3	78.7	83	87.3	0.729
	480	0.5	0.45	7.1	6.7	6.3	141.2-158.9	141.2	150.05	158.9	0.735
	480	0.25	0.7	7	6.7	6.4	143.6-156.7	143.6	150.15	156.7	0.768
	480	0.5	0.7	6.8	6.1	5.5	147.5-181.1	147.5	164.3	181.1	0.740
	480	0.25	0.45	6.8	6.6	6.3	146-159.1	146	152.55	159.1	0.765
	640	0.5	0.45	4.1	4	4	244.5-252	244.5	248.25	252	0.733
	640	0.25	0.7	4	4	3.9	247.1-256.4	247.1	251.75	256.4	0.770
	640	0.5	0.7	4.2	3.9	3.7	240.7-271.1	240.7	255.9	271.1	0.734
	640	0.25	0.45	4.2	4.1	3.9	238.1-254.4	238.1	246.25	254.4	0.771
YOLOv8l	320	0.5	0.45	7.3	7.1	6.9	136.9-145.3	136.9	141.1	145.3	0.715
	320	0.25	0.7	7.9	7.8	7.7	127.2-130.2	127.2	128.7	130.2	0.774
	320	0.5	0.7	7.7	7.1	6.7	130.7-149.3	130.7	140	149.3	0.716
	320	0.25	0.45	7.9	7.7	7.5	126.7-133.6	126.7	130.15	133.6	0.772
	480	0.5	0.45	3.8	3.7	3.7	264.9-270.7	264.9	267.8	270.7	0.769
	480	0.25	0.7	3.8	3.8	3.7	264.0-268.9	264	266.45	268.9	0.794
	480	0.5	0.7	3.8	3.6	3.4	262.2-296	262.2	279.1	296	0.772

	480	0.25	0.45	3.8	3.6	3.4	262.1-292	262.1	277.05	292	0.792
	640	0.5	0.45	2.6	2.4	2.3	390-431	390	410.5	431	0.762
	640	0.25	0.7	2.6	2.4	2.3	390.5-441.7	390.5	416.1	441.7	0.796
	640	0.5	0.7	2.5	2.3	2.2	398.9-462	398.9	430.45	462	0.767
	640	0.25	0.45	2.5	2.5	2.5	396.3-406.5	396.3	401.4	406.5	0.792

Table 4: YOLO modules performance on an AMD Ryzen 7 6500H CPU

Without acceleration, the fps for both YOLOv7 and YOLOv8 are not very impressive. The mean fps of each test case is colored by the colors red and green. In theory, a modern camera records at a rate of 30 fps, similar to a human eye that can recognize and see at 30-60 fps. So to get live (real-time) detection, the models have to reach a study rate of 30 fps or above. The inference time range is recorded in a Google Sheet and then calculated to get the mean fps value. The mean fps above 30 will be colored green with different intensity to indicate how fast can the model performs. On the other hand, red represents the mean fps values under 30, with the color intensifying as it approaches 0.

Having detection running under 30 fps is still acceptable for certain applications, where it is not essential for the module to detect everything but is expected to correctly identify objects. For instance, a system to detect traffic jams, with detections running as low as 1 fps, at any point where there are more vehicles than the threshold, with confidence, a traffic jam is detected. This is why it is important to test and log the fps and mAP of a module for different uses.

Next to consider is the performance of the YOLO models with speeding up using GPU.

4.2.2 Laptop GPU test

Configurations				**fps**			**Inference Time (ms)**				**mAP**
YOLO	**images size (px)**	**confidence**	**IoU**	**Min**	**Mean**	**Max**	**Range**	**Min**	**Mean**	**Max**	**mAP**
YOLOv7-tiny	320	0.5	0.45	140.8	125.8	113.6	7.1-8.8	7.1	7.95	8.8	0.352
	320	0.25	0.75	131.6	125	119	7.6-8.4	7.6	8	8.4	0.356
	320	0.5	0.75	140.8	129	119	7.1-8.4	7.1	7.75	8.4	0.354
	320	0.25	0.45	137	133.3	129.9	7.3-7.7	7.3	7.5	7.7	0.355
	480	0.5	0.45	131.6	122	113.6	7.6-8.8	7.6	8.2	8.8	0.364
	480	0.25	0.75	128.2	105.8	90.1	7.8-11.1	7.8	9.45	11.1	0.366
	480	0.5	0.75	140.8	137.9	135.1	7.1-7.4	7.1	7.25	7.4	0.364
	480	0.25	0.45	119	112.4	106.4	8.4-9.4	8.4	8.9	9.4	0.365
	640	0.5	0.45	142.9	134.2	126.6	7-7.9	7	7.45	7.9	0.368
	640	0.25	0.75	133.3	131.6	129.9	7.5-7.7	7.5	7.6	7.7	0.367
	640	0.5	0.75	140.8	135.1	129.9	7.1-7.7	7.1	7.4	7.7	0.370
	640	0.25	0.45	125	115.6	107.5	8-9.3	8	8.65	9.3	0.368
YOLOv7	320	0.5	0.45	89.3	86.6	84	11.2-11.9	11.2	11.55	11.9	0.485
	320	0.25	0.75	89.3	59.2	44.2	11.2-22.6	11.2	16.9	22.6	0.484
	320	0.5	0.75	87	56.3	41.7	11.5-24	11.5	17.75	24	0.487
	320	0.25	0.45	87.7	83.3	79.4	11.4-12.6	11.4	12	12.6	0.486
	480	0.5	0.45	78.1	41.2	7.9	12.8-35.8	12.8	24.3	35.8	0.496
	480	0.25	0.75	74.1	44.7	32.1	13.5-31.2	13.5	22.35	31.2	0.497

	480	0.5	0.75	73	49.1	37	13.7-27	13.7	20.35	27	0.499
	480	0.25	0.45	71.4	45.8	33.7	14-29.7	14	21.85	29.7	0.497
	640	0.5	0.45	69	38.2	26.5	14.5-37.8	14.5	26.15	37.8	0.511
	640	0.25	0.75	70.9	69	67.1	14.1-14.9	14.1	14.5	14.9	0.510
	640	0.5	0.75	70.9	43.5	31.3	14.1-31.9	14.1	23	31.9	0.512
	640	0.25	0.45	71.4	41.8	29.6	14-33.8	14	23.9	33.8	0.511
YOLOv7-e6	320	0.5	0.45	49.5	39	32.2	20.2-31.1	20.2	25.65	31.1	0.540
	320	0.25	0.75	49.8	47.4	45.2	20.1-22.1	20.1	21.1	22.1	0.538
	320	0.5	0.75	54.9	35.6	26.3	18.2-38	18.2	28.1	38	0.541
	320	0.25	0.45	54.3	52.1	50	18.4-20	18.4	19.2	20	0.540
	448	0.5	0.45	55.6	54.1	52.6	18-19	18	18.5	19	0.553
	448	0.25	0.75	55.6	53.1	50.8	18-19.7	18	18.85	19.7	0.552
	448	0.5	0.75	55.6	54.1	52.6	18-19	18	18.5	19	0.555
	448	0.25	0.45	55.6	53.6	51.8	18-19.3	18	18.65	19.3	0.554
	640	0.5	0.45	49.3	43.1	38.3	20.3-26.1	20.3	23.2	26.1	0.558
	640	0.25	0.75	50	50	50	20-20	20	20	20	0.557
	640	0.5	0.75	55.2	54.6	54.1	18.1-18.5	18.1	18.3	18.5	0.559
	640	0.25	0.45	54.9	54.2	53.5	18.2-18.7	18.2	18.45	18.7	0.559
YOLOv8n	320	0.5	0.45	122	113.6	106.4	8.2-9.4	8.2	8.8	9.4	0.486
	320	0.25	0.7	122	114.9	108.7	8.2-9.2	8.2	8.7	9.2	0.527
	320	0.5	0.7	120.5	114.9	109.9	8.3-9.1	8.3	8.7	9.1	0.486
	320	0.25	0.45	97.1	73.5	59.2	10.3-16.9	10.3	13.6	16.9	0.536
	480	0.5	0.45	111.1	103.6	97.1	9.0-10.3	9	9.65	10.3	0.555
	480	0.25	0.7	101	74.1	58.5	9.9-17.1	9.9	13.5	17.1	0.632
	480	0.5	0.7	108.7	101.5	95.2	9.2-10.5	9.2	9.85	10.5	0.555
	480	0.25	0.45	116.3	111.1	106.4	8.6-9.4	8.6	9	9.4	0.639
	640	0.5	0.45	103.1	72.7	56.2	9.7-17.8	9.7	13.75	17.8	0.564
	640	0.25	0.7	77.5	70.7	64.9	12.9-15.4	12.9	14.15	15.4	0.615
	640	0.5	0.7	113.6	109.3	105.3	8.8-9.5	8.8	9.15	9.5	0.560
	640	0.25	0.45	109.9	101	93.5	9.1-10.7	9.1	9.9	10.7	0.626
YOLOv8m	320	0.5	0.45	67.1	59.9	54.1	14.9-18.5	14.9	16.7	18.5	0.686
	320	0.25	0.7	71.4	66.7	62.5	14-16	14	15	16	0.727
	320	0.5	0.7	71.4	62.5	55.6	14-18	14	16	18	0.686
	320	0.25	0.45	66.7	64.5	62.5	15-16	15	15.5	16	0.729
	480	0.5	0.45	46.3	44.8	43.5	21.6-23	21.6	22.3	23	0.735
	480	0.25	0.7	47.4	45.9	44.4	21.1-22.5	21.1	21.8	22.5	0.768
	480	0.5	0.7	46.5	43.4	40.7	21.5-24.6	21.5	23.05	24.6	0.740
	480	0.25	0.45	45.7	44.5	43.5	21.9-23	21.9	22.45	23	0.765
	640	0.5	0.45	50	48.1	46.3	20-21.6	20	20.8	21.6	0.733
	640	0.25	0.7	51.5	49.4	47.4	19.4-21.1	19.4	20.25	21.1	0.770
	640	0.5	0.7	50	48.1	46.3	20-21.6	20	20.8	21.6	0.734
	640	0.25	0.45	50.5	48.2	46.1	19.8-21.7	19.8	20.75	21.7	0.771
YOLOv8l	320	0.5	0.45	55.6	52.6	50	18-20	18	19	20	0.715
	320	0.25	0.7	55.6	54.1	52.6	18-19	18	18.5	19	0.774
	320	0.5	0.7	54.1	49.4	45.5	18.5-22	18.5	20.25	22	0.716
	320	0.25	0.45	57.1	54.1	51.3	17.5-19.5	17.5	18.5	19.5	0.772
	480	0.5	0.45	50	44.9	40.8	20-24.5	20	22.25	24.5	0.769
	480	0.25	0.7	50	46.4	43.3	20-23.1	20	21.55	23.1	0.794

	480	0.5	0.7	48.3	46.4	44.6	20.7-22.4	20.7	21.55	22.4	0.772
	480	0.25	0.45	48.8	44.7	41.3	20.5-24.2	20.5	22.35	24.2	0.792
	640	0.5	0.45	35.5	34.6	33.8	28.2-29.6	28.2	28.9	29.6	0.762
	640	0.25	0.7	36.4	35	33.7	27.5-29.7	27.5	28.6	29.7	0.796
	640	0.5	0.7	35.6	35	34.4	28.1-29.1	28.1	28.6	29.1	0.767
	640	0.25	0.45	36.2	35.5	34.8	27.6-28.7	27.6	28.15	28.7	0.792

Table 5: YOLO modules performance accelerated by an NVIDIA GeForce RTX 3050 GPU

The performance of both YOLOv7 and YOLOv8 on a modern laptop GPU is outstanding. Running on the NVIDIA GeForce RTX 3050 GPU has produced results that are all above 30 fps, accommodating even the largest YOLO model can offer.

Generally, the values follow a similar trend, increasing the model weights and image size will produce more accurate detection results and a lower frame rate to compensate. The variety of mAP values is the result of different confidence and IoU threshold settings.

The confidence threshold determines the minimum confidence score required for a predicted bounding box to be considered valid. Lowering the confidence threshold can result in more bounding boxes being detected, including those with lower confidence scores. This can increase the number of True Positives but may also introduce more False Positives. As a result, it can impact both precision and recall, which are crucial components of mAP calculation.

The same can be said about the IoU threshold. Increasing the IoU threshold can make detections stricter by requiring a higher degree of overlap. This can lead to fewer True Positives being detected but also fewer False Positives.

Finally, the following table contains the performance comparison running on the Jetson Nano.

4.2.3 Jetson Nano test

Configurations				**fps**			**Inference Time (ms)**				**mAP**
YOLO	**images size (px)**	**confidence**	**IoU**	**Min**	**Mean**	**Max**	**Range**	**Min**	**Mean**	**Max**	**mAP**
YOLOv7-tiny	320	0.5	0.45	21.9	21.8	21.7	45.6-46	45.6	45.8	46	0.352
	320	0.25	0.75	21.7	21.7	21.7	46-46	46	46	46	0.356
	320	0.5	0.75	21.7	21.7	21.7	46-46	46	46	46	0.354
	320	0.25	0.45	21.7	21.4	21	46-47.6	46	46.8	47.6	0.355
	480	0.5	0.45	13.2	13.2	13.2	76-76	76	76	76	0.364
	480	0.25	0.75	12.7	12.7	12.7	79-79	79	79	79	0.366
	480	0.5	0.75	13.2	13.2	13.2	76-76	76	76	76	0.364
	480	0.25	0.45	12.7	12.5	12.3	79-81	79	80	81	0.365
	640	0.5	0.45	8.9	8.9	8.8	112-113	112	112.5	113	0.368
	640	0.25	0.75	8.8	8.8	8.8	113-114	113	113.5	114	0.367
	640	0.5	0.75	8.8	8.8	8.7	113-115	113	114	115	0.370
	640	0.25	0.45	9	8.9	8.8	111-114	111	112.5	114	0.368
	320	0.5	0.45	5	5	5	199-200	199	199.5	200	0.485
	320	0.25	0.75	4.9	4.9	4.9	203-204	203	203.5	204	0.484

YOLOv7	320	0.5	0.75	5	4.9	4.9	202-205	202	203.5	205	0.487
	320	0.25	0.45	5.3	5.1	5	190-200	190	195	200	0.486
	480	0.5	0.45	2.9	2.9	2.9	347-348	347	347.5	348	0.496
	480	0.25	0.75	2.9	2.9	2.9	346-347	346	346.5	347	0.497
	480	0.5	0.75	2.9	2.9	2.9	346-347	346	346.5	347	0.499
	480	0.25	0.45	2.9	2.9	2.9	346-347	346	346.5	347	0.497
	640	0.5	0.45	1.9	1.9	1.9	537-540	537	538.5	540	0.511
	640	0.25	0.75	1.9	1.9	1.9	538-539	538	538.5	539	0.510
	640	0.5	0.75	1.9	1.9	1.9	538-540	538	539	540	0.512
	640	0.25	0.45	1.9	1.9	1.9	537-538	537	537.5	538	0.511
YOLOv7-e6	320	0.5	0.45	2.9	2.9	2.9	343-350	343	346.5	350	0.540
	320	0.25	0.75	2.9	2.9	2.8	344-351	344	347.5	351	0.538
	320	0.5	0.75	3.1	3	2.8	320-356	320	338	356	0.541
	320	0.25	0.45	2.9	2.9	2.8	342-358	342	350	358	0.540
	448	0.5	0.45	1.8	1.8	1.7	560-573	560	566.5	573	0.553
	448	0.25	0.75	1.8	1.8	1.8	543-560	543	551.5	560	0.552
	448	0.5	0.75	1.8	1.8	1.7	554-572	554	563	572	0.555
	448	0.25	0.45	1.8	1.8	1.8	559-567	559	563	567	0.554
	640	0.5	0.45	1	0.7	0.5	1050-2006	1050	1528	2006	0.558
	640	0.25	0.75	0.9	0.7	0.5	1082-1938	1082	1510	1938	0.557
	640	0.5	0.75	0.9	0.7	0.5	1176-1820	1176	1498	1820	0.559
	640	0.25	0.45	0.7	0.6	0.5	1469-1907	1469	1688	1907	0.559
YOLOv8n	320	0.5	0.45	23.8	23.5	23.3	42-43	42	42.5	43	0.486
	320	0.25	0.7	23.8	23.5	23.3	42-43	42	42.5	43	0.527
	320	0.5	0.7	23.8	23.5	23.3	42-43	42	42.5	43	0.486
	320	0.25	0.45	24.4	23.5	22.7	41-44	41	42.5	44	0.536
	480	0.5	0.45	11.5	11.4	11.4	87-88	87	87.5	88	0.555
	480	0.25	0.7	11.5	11.4	11.2	87-89	87	88	89	0.632
	480	0.5	0.7	11.8	11.6	11.5	85-87	85	86	87	0.555
	480	0.25	0.45	11.8	11.6	11.5	85-87	85	86	87	0.639
	640	0.5	0.45	7.4	7.3	7.2	136-139	136	137.5	139	0.564
	640	0.25	0.7	7.4	7.4	7.3	135-137	135	136	137	0.615
	640	0.5	0.7	7.4	7.3	7.2	135-139	135	137	139	0.560
	640	0.25	0.45	7.4	7.3	7.2	136-138	136	137	138	0.626
YOLOv8m	320	0.5	0.45	11.8	11.4	11.1	85-90	85	87.5	90	0.686
	320	0.25	0.7	11.9	11.6	11.2	84-89	84	86.5	89	0.727
	320	0.5	0.7	11.9	11.3	10.8	84-93	84	88.5	93	0.686
	320	0.25	0.45	11.6	11.4	11.2	86-89	86	87.5	89	0.729
	480	0.5	0.45	7.2	7.2	7.1	138-141	138	139.5	141	0.735
	480	0.25	0.7	7.3	7.2	7	137-142	137	139.5	142	0.768
	480	0.5	0.7	7.4	7.3	7.2	136-139	136	137.5	139	0.740
	480	0.25	0.45	7.2	7.2	7.1	138-140	138	139	140	0.765
	640	0.5	0.45	2.8	2.7	2.6	362-385	362	373.5	385	0.733
	640	0.25	0.7	2.9	2.8	2.8	349-361	349	355	361	0.770
	640	0.5	0.7	2.8	2.8	2.7	351-372	351	361.5	372	0.734
	640	0.25	0.45	2.9	2.8	2.8	348-358	348	353	358	0.771
	320	0.5	0.45	7.4	7.2	7.1	135-141	135	138	141	0.715
	320	0.25	0.7	7.5	7.2	7	134-143	134	138.5	143	0.774
	320	0.5	0.7	7.8	7.4	7.1	129-140	129	134.5	140	0.716

YOLOv8l	320	0.25	0.45	7.6	7.2	6.9	131-145	131	138	145	0.772
	480	0.5	0.45	2.8	2.7	2.6	357-379	357	368	379	0.769
	480	0.25	0.7	2.7	2.7	2.6	368-380	368	374	380	0.794
	480	0.5	0.7	2.7	2.7	2.6	370-381	370	375.5	381	0.772
	480	0.25	0.45	2.6	2.6	2.6	378-386	378	382	386	0.792
	640	0.5	0.45	1.7	1.7	1.6	587-608	587	597.5	608	0.762
	640	0.25	0.7	1.7	1.7	1.7	579-596	579	587.5	596	0.796
	640	0.5	0.7	1.7	1.7	1.7	574-585	574	579.5	585	0.767
	640	0.25	0.45	1.8	1.7	1.7	569-587	569	578	587	0.792

Table 6: YOLO modules performance on the Jetson Nano

The performance of the Jetson Nano aligns closely with expectations in the context of its role as an edge computing device. Among the various models in the Jetson lineup, the Nano, being the smallest, offers a well-balanced combination of processing power and size. When considering the different YOLO versions, YOLOv8 emerges as the clear leader, outpacing YOLOv7 in terms of both frames per second (fps) and mean average precision (mAP). This reinforces the notion that the latest advancements in the YOLO series contribute to improved detection accuracy and speed.

However, it's noteworthy to emphasize that the suitability of certain YOLO module weights varies based on the application's real-time requirements. The medium and large weights of the YOLO modules may not be the most optimal choices for real-time detections on the Jetson Nano due to the computational demands they place on the hardware. These weights could lead to reduced performance and potentially compromise the desired real-time detection capability.

In contrast, the performance of YOLO's tiny weights, utilizing an image size of 320 pixels, aligns more closely with the Jetson Nano's capabilities. This configuration offers a closer approximation to achieving the target of 30 frames per second. The careful consideration of both model version and weight choice is essential when aiming to strike the right balance between accuracy and speed on edge devices like the Jetson Nano.

Interestingly, across three tests, despite using different CPUs, the mAP stays consistent across the board.

4.3 Conclusion

It's worth noting that while the Jetson Nano offers commendable performance, modern laptop GPUs have demonstrated superior performance, outperforming the Nano in terms of both frame rate and accuracy. Additionally, when comparing the Jetson Nano's performance against CPUs, the difference in capability is noticeable but not substantial. However, the laptop should not be placed in the edge device position for the reasons mentioned in previous sections. In conclusion, the Jetson Nano effectively fulfills its role as an edge computing solution, with YOLOv8 standing out as the preferred option due to its advanced capabilities.

5 YOLOv8 tracking and logging

Having decided on the model to be used for the project, this section contends with the application of YOLOv8 in a meaningful way.

5.1 Tracking

5.1.1 Overview

Taking the original vision for the project, the system with the assistance of YOLOv8 has to identify the defective items on a production line. To do so, the item being detected is then distinguished from other objects. A solution to this problem is to give the objects identification numbers. This is easier said than done, however, due to the nature of YOLO or object detection in general. YOLO processes a video feed from a camera frame by frame, detecting every object in a frame and then notating it for the user to see. The question arose: How can the machine recognize 2 detections on different frames are the same object?

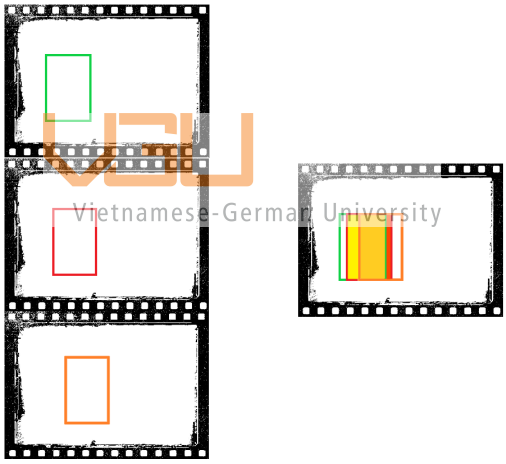
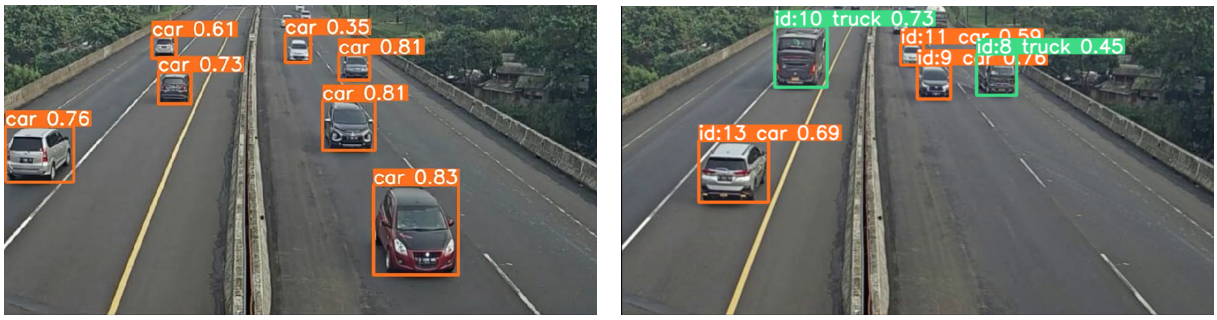


Figure 8: How to recognize the same object across frames

In the simplest terms possible, the same object should not move too far from its position in the previous frame. By comparing the position of the box notations between frames, there is an inevitable overlap of the same object. This could also lead to miss identification as a different object can move and overlap with another object's position. Setting an intersection over union value can help negate these situations.

5.1.2 YOLOv8 built-in tracking function

The choice of YOLOv8 offers the benefit of a built-in tracking application. Ultralytics YOLO supports the following tracking algorithms: BoT-SORT and ByteTrack. In addition, Ultralytics allows for the use of a custom tracker by including the YAML file as an argument. However, this thesis will only focus on the 2 offered trackers for simplicity.



(a) YOLOv8 detect

(b) YOLOv8 track

Figure 9: YOLOv8 object detection and YOLOv8 object detection with tracking

a) BoT-SORT

BoT-SORT is the latest in a long line of SORT-like (Simple Online and Realtime Tracking) algorithms. Multi-object tracking (MOT) is the field of detecting and tracking all the objects in a scene while maintaining a unique identifier for each object. Introduced in mid-2022, BoT-SORT is a state-of-the-art tracker and a solution for the MOT problem.

An over-simplified SORT tracking algorithm is broken down into these steps: In the first step, for each frame of a video, an object detection algorithm identifies and localizes objects of interest; in this case, the source is the YOLOv8 model. Next, the algorithm associates detections from the current frame with existing tracked objects from the previous frames. This association is typically based on metrics like the Euclidean distance between detection bounding boxes and predicted object positions. Tracked objects have a predicted position and velocity based on their previous states. The SORT algorithm predicts the next position of each tracked object based on its current state. The primary challenge in multi-object tracking is associating detections with existing tracks. SORT uses techniques, like in the case of BoT-SORT, Kalman filtering, to optimally associate detections and tracks based on the similarity of their properties. After association, tracked object states are updated based on new detections. This includes adjusting the position, velocity, and other attributes of the tracked objects. SORT also handles the creation of new tracks for detections that don't match any existing tracks and manages the termination of tracks for objects that are no longer detected.

From there, BoT-SORT is introduced with further improvements, including a camera motion compensation-based features tracker to improve the accuracy of box localization. This tracker compensates for the camera's motion to ensure that the features used for tracking are consistent across frames. Using a new method for IoU and ReID's cosine-distance fusion to improve the robustness of associations between detections and tracks. This method fuses the IoU and ReID cosine distance scores to obtain a more reliable association score. Finally, integrates the limitations of existing SORT-like trackers into the novel ByteTrack to address these limitations and improve the tracker's performance. These advancements have led to BoT-SORT outperforming other tracking algorithms on the MOT17 and MOT20 challenges, making it a promising multi-pedestrian tracking algorithm.

b) ByteTrack

The other default tracker for YOLOv8 is ByteTrack. Introduced in 2021, it is a milestone in the MOT algorithms. It also made up a part of BoT-SORT as mentioned before. At the time, ByteTrack demonstrated superior results over existing trackers while maintaining steady 30 fps on a single V100 GPU.

The accompanying paper for ByteTrack stated that it achieved state-of-the-art performance on multiple tracking benchmarks, including MOT17, MOT20, HiEve, and BDD100K. Its results were only passed by the BoT-SORT a year later.

The proposed method, BYTE, is a simple, effective, and generic association method for multi-object tracking. It associates almost every detection box instead of only high-score ones to recover true objects and filter out background detections. The method consists of two modes: The first mode associates detection boxes with tracklets based on their Re-ID features and motion information. This mode is used for high-score detection boxes; The second mode associates detection boxes with tracklets based on their similarity in appearance and motion. This mode is used for low-score detection boxes.

However, the main draw for ByteTrack is its high accuracy, low inference time, and flexibility to integrate and improve on other association methods. The results show that it improves the performance of multi-object tracking, especially for low-score detection boxes. By applying to existing trackers, they achieve consistent improvements in tracking performance.

5.1.3 Conclusion

While ByteTrack provided many improvements and advantages at the time, with still the landmark for other MOT algorithms to follow. BoT-SORT outperforms ByteTrack in every aspect. This is easy to understand as BoT-SORT took ByteTrack as its base and improved upon its weaknesses. Under certain circumstances, ByteTrack may prove to be more suitable, but for the purpose and the original idea of the project, BoT-SORT is chosen to be the tracking algorithm for YOLOv8.

5.2 Logging

5.2.1 Overview

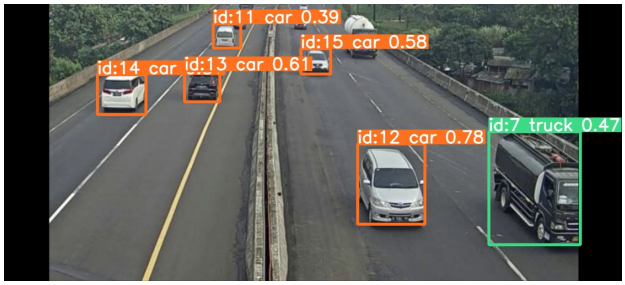
With the detections of YOLOv8 with tracking established, the data collected will be useless if it is not logged out or processed. Once YOLOv8 detections with tracking capabilities are established, harnessing the potential of the collected data becomes paramount. The efficacy of the detections would be severely compromised if this valuable data is not adequately managed through proper logging and processing mechanisms. The process of logging and processing serves as the bridge between real-time detection and actionable insights, ensuring that the entire endeavor is not merely an exercise in capturing information but a transformative journey toward informed decision-making. This task is managed by Supervision.

5.2.2 Supervision

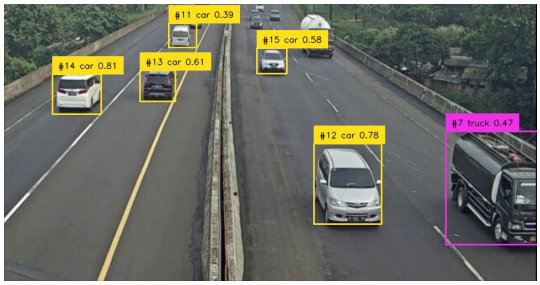
Figure 10: Supervision counting objects example

Supervision, an impressive assortment of computer vision tools, has emerged as a game-changing creation from the innovation labs of Roboflow. This meticulously designed Python extension has been tailored to elevate the capabilities of computer vision projects to new heights, bringing a wealth of advanced functionalities to the fingertips of developers and researchers alike. With Supervision, the realm of computer vision is no longer confined to basic tasks; it now offers a multifaceted toolkit that empowers professionals to achieve more, streamline workflows, and extract valuable insights from their visual data. However, the application of focus for this project is counting objects in certain areas and how it is repurposed.

a) How to extract data from YOLOv8



(a) YOLOv8



(b) YOLOv8 + Supervision

Figure 11: Translating data from YOLOv8 to Supervision

To establish a seamless integration between YOLOv8 and Supervision, a crucial step entails the extraction and translation of data detected by YOLOv8 to a format compatible with the Supervision platform. This step serves as the linchpin for enabling Supervision's object-counting functionality to leverage the data extracted from YOLOv8, thereby facilitating precise and insightful calculations.

The function at the core of this data translation process is *sv.Detections.from_ ultralytics(result)*. This ingenious function serves as the bridge that connects the outcomes of YOLOv8's object detection efforts with the Supervision environment. It plays a pivotal role in transferring key information such as *confidence*, *class_ id*, and *tracker_ id* associated with the detected bounding boxes from YOLOv8's outputs to Supervision's data pipeline.

Notably, the function doesn't stop at merely transferring essential detection attributes; it goes the extra mile by ensuring that the spatial context of the detected objects is meticulously preserved. This is accomplished by including the precise coordinates of the bounding boxes in the transferred data. These coordinates serve as the foundation for the annotation process in Supervision, allowing for the seamless visualization of detected objects within the Supervision interface.

b) Introducing Supervision counting line

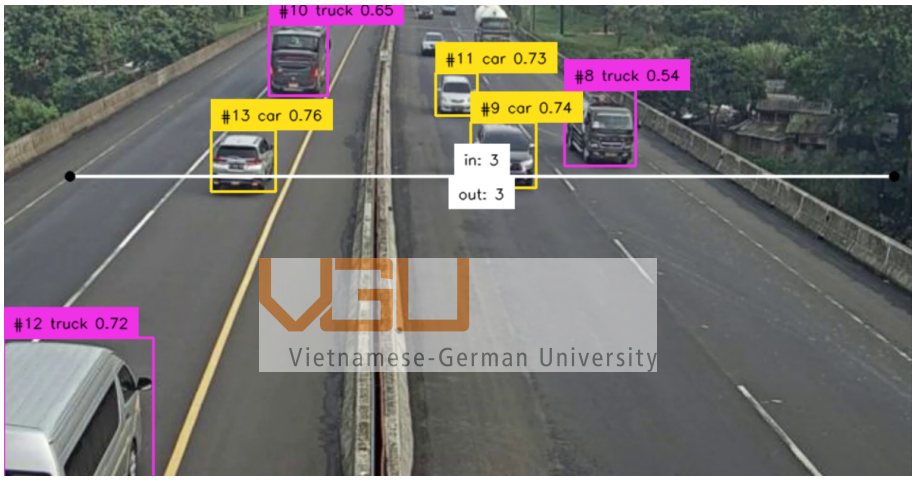


Figure 12: Applying counting line to the project

Once the data stream from YOLOv8 has been harnessed, the process of integrating counting lines into projects within the Supervision framework is remarkably straightforward. This entails a user-friendly procedure that efficiently augments the data analysis capabilities. By designating two pivotal points, Point A and Point B, Supervision promptly crafts a *LineZone* that effectively links these two coordinates. This strategic maneuver orchestrates the establishment of a virtual line of demarcation, ready to interact with the detected objects.

However, it's important to note that the implementation of this *LineZone* extends beyond a mere visual representation. While Point A and Point B provide the foundational context for the line's placement, the bar itself effectively extends beyond these designated coordinates, spanning across the entirety of the frame. This robust approach ensures that no subtlety of object interaction goes unnoticed, even when objects engage with the line in ways that transcend the initial two points.

The synergy between the bounding boxes derived from YOLOv8 detections and the newly minted counting line is a multi-dimensional engagement that unfolds in distinct phases. Firstly, there are new detections that materialize as objects make their appearance. These objects, once detected

on one side of the line, initiate their journey through the data zone.

Secondly, the line interacts with objects that are mid-transit, caught in the throes of traversing the visual field. This dynamic interaction is captured as the boxes, indicative of the detected objects, occupy a middle-ground state within the line zone.

Lastly, the objects that have successfully crossed the line and moved to the other side are met with precision tracking. This milestone is tactfully leveraged by Supervision's *in* and *out* counters, which meticulously tally the objects based on their directional movement about the line.

Nevertheless, there exist certain scenarios where the initiation of the counter fails to transpire seamlessly. These instances are pivotal in highlighting the intricacies of the Supervision-YOLOv8 synergy. One such occurrence arises when detections or boxes first manifest themselves directly on the line. This predicament ushers in a confounding challenge, as the program grapples with the dilemma of assigning these objects to a specific side of the line for counting.

In a similar vein, a second scenario emerges when YOLOv8 temporarily loses its grasp on object tracking during the critical juncture of crossing the line. This predicament materializes when an object is traversing the line and YOLOv8 momentarily loses its ability to sustain accurate tracking. This transient disconnect between the tracking mechanism and the object's movement across the line disrupts the counting process, leading to discrepancies in the final count.

c) The nature of LineZone's points

Integrating logging functionality through the utilization of the *LineZone* entails a strategic approach that involves overriding key functions. This method allows for a seamless fusion between the Supervision platform and the distinctive attributes of the *LineZone*. To embark on this endeavor, it is imperative to delve into the fundamental characteristics of the points employed to delineate the *LineZone*. These points serve a dual purpose: not only do they facilitate the fine-tuning of the optimal angle for object passage, but they also establish a categorical distinction between the two sides of the line.

To clarify this distinction, let's explore the concept through a simple visual analogy depicted in Figure 13. Imagine your left hand positioned horizontally, with your thumb pointing towards your body. In this setup, we will be using your hand as a model for the *LineZone*, where point A corresponds to your wrist, and point B corresponds to the tip of your finger.

In this hand analogy, the *LineZone* operates as follows: As objects traverse through the line, their movement direction determines whether the *in* or *out* counter is affected. If an object moves through by approaching from the back of the hand, it will increment the *out* counter. Conversely, if an object moves towards the palm of your hand, it will increment the *in* counter.

Now, let's consider the scenario when you move your hand while maintaining the same thumb orientation facing your body. Despite the movement, points A and B still correspond to the wrist and the tip of the fingers, respectively. However, due to the hand's relocation, the sides of the line also shift accordingly.

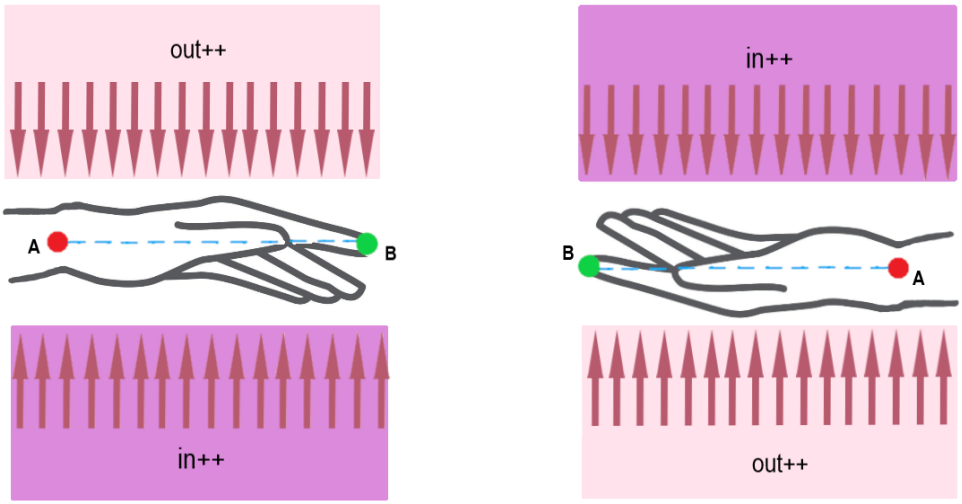


Figure 13: Visualization of how points in *LineZone* work

Crucially, what doesn't change is the relationship between the object's movement and the counters. Irrespective of the hand's position, an object moving towards the palm will consistently increase the *in* counter, while an object moving towards the back of the hand will consistently boost the *out* counter.

d) Overriding the LineZone trigger function

The function responsible for counting objects crossing the *LineZone* is the *trigger* function. This function efficiently manages the various states an object assumes within the context of the *LineZone*, as elaborated in the preceding sections. The motivation behind overriding this function lies in the desire to introduce additional steps for gathering information based on the specific states an object undergoes.

As elucidated in the section titled **The Nature of LineZone's Points**, the side of the line can be predetermined by manipulating the two input points that define the *LineZone*. This insight provides the groundwork for bifurcating the *trigger* function into two distinct functions: *trigger1* and *trigger2*. Each of these functions serves a unique purpose, catering to the different behavior of objects concerning the *in* and *out* counters, respectively.

To delve deeper into the specifics, let's consider the *trigger1* function. This function addresses the border that an object must cross to enter a zone. So the sole purpose of this function is the initialize the *time_ in* (the time when the object first entered the area).

Listing 1: *LineZone* trigger1 for objects to enter

```
def trigger1(self, detections: Detections):
    for xyxy, _, confidence, class_id, tracker_id in detections:
        # unchanged code
        ...

        if tracker_state:
            self.in_count += 1
        else:
            self.out_count += 1
            if tracker_id in my_dict:
                my_dict.get(tracker_id).set_time_in(time.time())
```

On the other hand, the *trigger2* function handles instances where objects move towards the side of the line that increases the *in* counter. In this context, modifications to the handle of new detections are changed to also initialize classes for logging the information. Specifically, it initializes the object *tracker_id*, the starting *class_id*, and assigns a UUID.

Handling of detection on the same side of *trigger2* is also subject to change to include a fail-safe. As long as the object remains on the same side, the *confidence* corresponds to the *class_id* are added to determine the definitive *class_id* of an object. This continues till finally, when the object crosses the line from the increment *in* side, the *time_out* (the time when the object leaves the zone) and the final result for *class_id* are logged and all the information is gathered.

Listing 2: *LineZone* trigger2 for objects to exit

```
def trigger2(self, detections: Detections):
    for xyxy, _, confidence, class_id, tracker_id in detections:
        # unchanged code
        ...

        # handle new detection
        if tracker_id not in self.tracker_state:
            self.tracker_state[tracker_id] = tracker_state
            my_dict[tracker_id] = collectInfo(tracker_id)
            continue

        # handle detection on the same side of the line
        if self.tracker_state.get(tracker_id) == tracker_state:
            if tracker_id in my_dict:
                my_dict[tracker_id].confirm_class_id(class_id,confidence)
            continue

        # unchanged code
        ...
```

```
if tracker_state:
    self.in_count += 1
    my_dict.get(tracker_id).set_time_out(time.time())
    my_dict.get(tracker_id).set_class_id()
    my_dict.get(tracker_id).post_info()
    del my_dict[tracker_id]

else:
    self.out_count += 1
```

5.2.3 Conclusion

By introducing the new trigger functions, the zone configuration is now encompassed by four distinct lines. This arrangement involves two lines dedicated to managing detections that are entering the designated area, and another two lines are assigned to oversee detections as they exit the same area. This symmetrical layout ensures that the movement of objects through the zone is accurately tracked and accounted for, both during their entry and exit phases.

Fine-tuning these four lines becomes crucial to optimizing the system's response to objects approaching or receding from the camera's POV. This adjustment accounts for scenarios where objects are either entering the frame for the first time or are positioned too distantly to be promptly detected. Given the constraints of the Jetson Nano's processing capabilities for near real-time detection, accurately identifying objects at the initial moments they appear within the frame, or when they are positioned at a significant distance, can be challenging. So the instance of tracking traffic on a highway, vehicles moving toward the camera need their lines pushed closer to the front. The opposite is true for traffic moving away from the camera as the lines tracking them need to be pushed back.

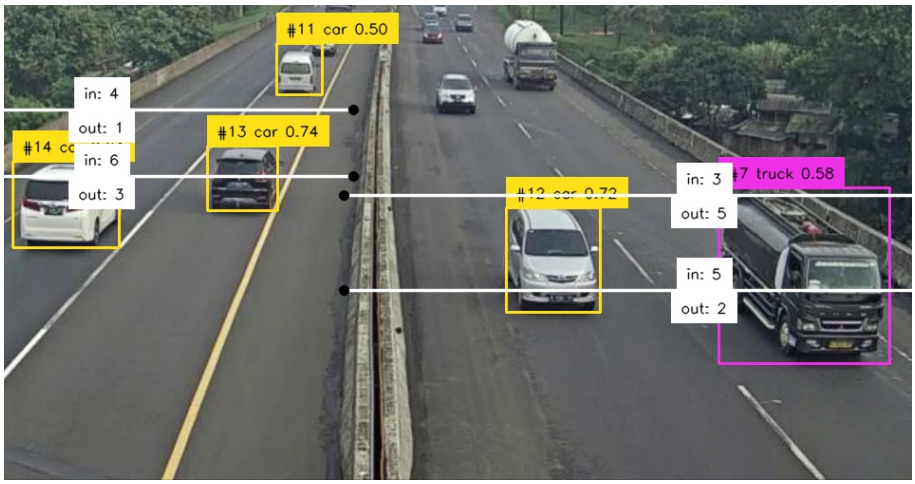


Figure 14: The modified *trigger* function implemented

6 Internet of Things of AIoT

This section discusses the IoT part of an AIoT system. This includes an overview of IoT's current climate and the project's IoT choices.

6.1 Overview

The Internet of Things has evolved into a transformative technology landscape, connecting devices, sensors, and systems to the Internet, enabling them to collect, exchange, and analyze data for various applications. The proliferation of IoT devices has brought about significant advancements across industries, revolutionizing sectors like healthcare, transportation, agriculture, smart cities, and more.

In the current IoT landscape, devices are embedded with sensors, processors, and communication capabilities, allowing them to communicate with each other and with central platforms or servers. This connectivity enables the real-time monitoring and control of remote devices and assets, leading to improved efficiency, data-driven insights, and enhanced decision-making.

6.2 IoT platform choices

For this project, there 2 main IoT platforms that are available for experiments and productions

6.2.1 Things Platform

Figure 15: ThingsBoard Cloud Platform

The rapid growth of the IoT has led to the development of advanced platforms that facilitate the management, monitoring, and analysis of connected devices and their data. One such notable platform is Thingsboard Cloud, an innovative IoT solution that empowers organizations to harness the potential of their IoT ecosystems. With a range of features and capabilities, Thingsboard Cloud offers a comprehensive suite of tools to seamlessly connect, manage, and extract insights from IoT devices.

Thingsboard Cloud serves as a robust and scalable IoT platform that caters to the diverse needs of industries ranging from manufacturing and agriculture to healthcare and smart cities. It provides a unified dashboard that enables users to visualize real-time data, track device status,

and analyze historical trends. This centralized view enhances operational efficiency and informed decision-making by providing a holistic understanding of the connected devices' performance.

Moreover, Thingsboard Cloud emphasizes security and data privacy. It offers robust authentication mechanisms and end-to-end encryption to safeguard sensitive data from unauthorized access. Additionally, the platform provides user access control, ensuring that only authorized personnel can view, manage, and manipulate IoT devices and their data.

Another advantage of Thingsboard Cloud is its analytics capabilities. The platform allows users to perform advanced data analysis, generate reports, and gain actionable insights from the collected IoT data. This empowers businesses to optimize processes, improve efficiency, and uncover hidden patterns that might not be immediately apparent.

Thingsboard represents more than just an IoT platform; it embodies the spirit of collaboration and innovation that open-source projects are known for. Built on a foundation of transparent development and community engagement, Thingsboard's open-source nature empowers developers, researchers, and enterprises to craft tailored IoT solutions that suit their specific needs. By providing access to its source code, the platform invites contributions from a diverse range of minds, fostering an environment of continuous improvement and customization.

An excellent demonstration of Thingsboard's open-source ethos can be witnessed through a notable instantiation called Things.vn. Notably, Things.vn caters specifically to the vibrant community of Vietnamese developers, accentuating the platform's commitment to localized and region-specific IoT solutions.

Figure 16: Things.vn Cloud Platform

Being an implementation of Thingsboard, Things.vn offers the same range of applications and a variety of tools to monitor, visualize, and analyze data as an IoT platform. With a wide range of industry-standard IoT communication protocols, such as MQTT(s), HTTP(s), and CoAP.

In addition, the Things.vn platform also provides services to set up smart applications based on consumer requests, from smart factories and workplaces to monitoring air quality and storage.

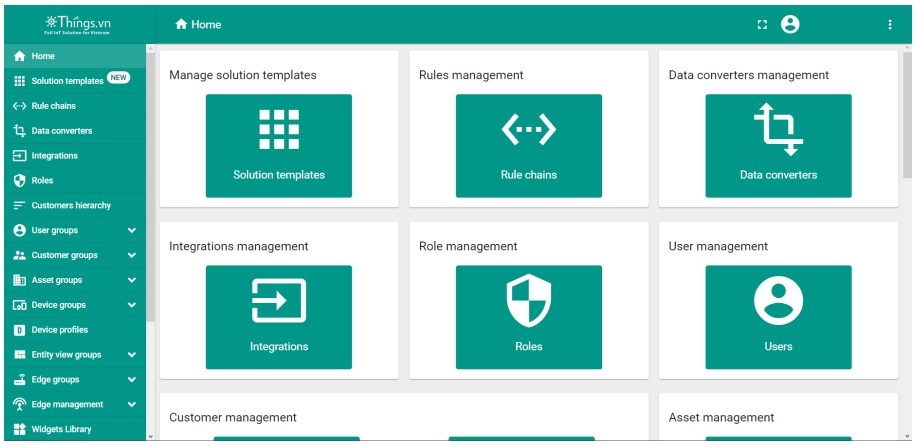


Figure 17: Things.vn Cloud Landing Page

6.2.2 InfluxDB

Figure 18: InfluxDB Platform

In the realm of the IoT, the effective management and analysis of time-series data is crucial. One prominent player in this field is InfluxDB, a powerful and open-source time-series database designed to handle the unique challenges posed by IoT data streams. InfluxDB has rapidly gained popularity as a solution for storing, querying, and visualizing time-stamped data, making it an essential tool for businesses seeking to harness the value of their IoT ecosystems.

At its core, InfluxDB excels in managing data points that are timestamped and associated with specific measurements. This aligns perfectly with the nature of IoT data, where sensors and devices generate continuous streams of time-sensitive information. InfluxDB's architecture allows for efficient storage and retrieval of this data, enabling users to manage massive amounts of information generated by countless IoT devices.

One of the key strengths of InfluxDB is its optimized querying capabilities for time-series data. The platform offers a query language, InfluxQL, designed to efficiently handle time-based data queries. This means that users can easily perform tasks such as filtering data within specific time intervals, aggregating data over time, and performing calculations on temporal data. Such capabilities are essential for extracting valuable insights from IoT data streams.

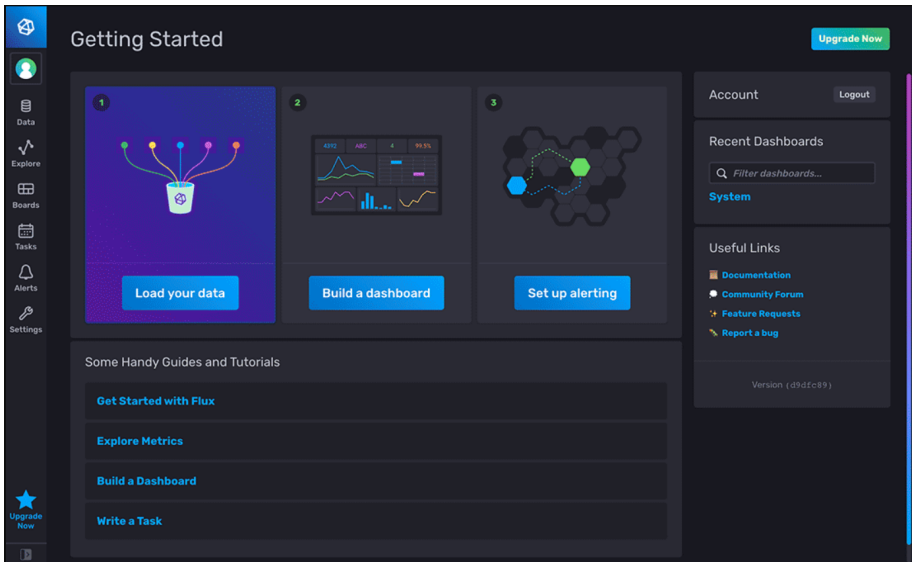


Figure 19: InfluxDB Platform Landing Page

InfluxDB also boasts a rich ecosystem of tools and integrations that enhance its usability. Grafana, for example, is a popular visualization platform that can be seamlessly integrated with InfluxDB to create visually appealing and informative dashboards. This integration empowers users to gain real-time insights into their IoT data, enabling them to monitor device status, detect anomalies, and make data-driven decisions.

Security is a top priority for any IoT system, and InfluxDB addresses this concern through features such as authentication, authorization, and encryption. This ensures that data remains protected throughout its lifecycle, from ingestion to analysis.

6.3 Conclusion

In the end, both platforms provide unique attributes that do not make one inherently better than the other. Things.vn excels as an end-to-end IoT platform, offering device management, data processing, visualization, and rule-based automation. InfluxDB specializes in efficiently storing and retrieving time-series data, making it a powerful backend choice for applications heavily reliant on historical data tracking and analysis. During this project, these two IoT platforms are both used interchangeably, with the usage of specific platform will be mentioned if necessary.

7 Miscellaneous tools and accessorries

In this section, the auxiliary tools and applications required for the system to function efficiently are discussed. These tools do not form the primary core of the project. Some temporary solutions discussed here were replaced by alternative solutions, which we will also address in this section.

7.1 Fluentd

Figure 20: An open source data collector - Fluentd

Fluentd is an open-source data collection tool designed to simplify and streamline the process of collecting, processing, and forwarding log data from various sources in a computing environment. It acts as a middleware, facilitating the integration and transportation of data between different applications and systems, which is particularly crucial in complex and distributed architectures.

The primary purpose of Fluentd is to address the challenges associated with log data management in modern IT environments. As systems and applications become more distributed and diverse, generating a significant volume of log data, the need for a centralized mechanism to collect and analyze this data efficiently has grown. Fluentd was developed to fill this gap by providing a unified platform to manage logs from a wide range of sources, such as servers, applications, databases, and external services.

Fluentd follows a "log forwarding" architecture, where it acts as a data collector that gathers logs from various sources and processes them before forwarding them to destinations such as databases, storage systems, or analytics platforms. Its flexibility lies in its ability to accommodate data in various formats and protocols, including JSON, CSV, and Apache logs, among others. This versatility allows Fluentd to work seamlessly in heterogeneous environments with diverse logging practices.

One of the significant advantages of Fluentd is its extensive ecosystem of plugins. These plugins enable seamless integration with various data sources, destinations, and processing tools, allowing users to tailor Fluentd to their specific needs. This extensibility has contributed to Fluentd's popularity in both small-scale setups and large, complex architectures.

In this project, Fluentd plays a crucial role in monitoring changes to logging files and transmit-

ting the collected data to the designated IoT database. Leveraging Fluentd's versatile plugins, and establishing a connection with InfluxDB becomes a straightforward process. In parallel, Things.vn utilizes an HTTP connection to relay telemetry data. Adjustments are made solely in the configuration file to instruct Fluentd to monitor the specific file, ensuring data integrity through backup mechanisms to prevent information loss or duplication.

Listing 3: Fluentd Configuration

```
<source>
    @type tail
    format json
    read_from_head true
    tag file-myapp.log
    path ./file/info.log
    pos_file ./tmp/info.log.pos
</source>

<match file-myapp.log>
    # Configuration for InfluxDB
    @type influxdb2
    url http://...
    org ...
    token ...
    bucket ...
    use_ssl false

    # Configuration for Things.vn
    @type HTTP

    endpoint http://cloud.things.vn/api/v1/.../telemetry
    open_timeout 2

    <format>
        @type json
    </format>

    # Configuration for both
    <buffer>
        flush_interval 1s
    </buffer>
</match>
```

7.2 ImageZMQ

ImageZMQ is a powerful and efficient communication library that simplifies the process of sending and receiving images between different devices and applications. Built on top of ZeroMQ, a high-performance messaging library, ImageZMQ specializes in the seamless transmission of images, making it an invaluable tool in various fields such as computer vision, machine learning, and IoT applications.

The core functionality of ImageZMQ revolves around the concept of image serialization and communication. It enables devices and processes to transmit images, frames, or arrays over a network with minimal latency and efficient data serialization. This is particularly useful in scenarios where real-time image data needs to be shared between different components of a system.

Originally developed to receive surveillance footage from Raspberry Pi cameras. During the early stages of the project, it was used to stream footage processed from the Jetson Nano to a streaming server to broadcast the video back to the users. By utilizing ZeroMQ's high-speed messaging patterns, ImageZMQ ensures that images can be transmitted quickly and reliably, even in resource-constrained environments.

However, ImageZMQ is proven to be only a temporary solution as it is not sufficient to handle multiple image channels because of its single IP address binding. As the project expanded, the feature of allowing the user to choose the mode of watching the stream playback was introduced, and ImageZMQ was proven obsolete.

7.3 Streamlit

Figure 21: A faster way to build and share data apps - Streamlit

To stream content to users, there needs to be a user interface through which they can view the footage. This is where Streamlit comes in handy. Streamlit, an open-source Python library, has emerged as a transformative tool for data scientists and developers seeking to convert data scripts into interactive web applications with remarkable ease. Its primary goal is to simplify the process of transforming data visualizations, machine learning models, and other data-centric scripts into shareable, responsive, and interactive web apps, all while minimizing the need for intricate web development expertise.

At the heart of Streamlit's appeal is its remarkable simplicity. By adopting a user-friendly syntax, the library empowers users to effortlessly incorporate interactive elements into their applications.

With a single line of code, interactive components such as sliders, buttons, and text inputs can be seamlessly integrated, resulting in a streamlined development experience.

While its simplicity is a hallmark, Streamlit also offers robust customization options. Users can fine-tune the appearance and behavior of their web apps by adjusting layouts, themes, and styles. This balance between ease of use and customization empowers developers to tailor their applications to meet specific requirements without compromising efficiency.

Despite its numerous advantages, it is important to acknowledge that Streamlit, like any technology, is not without its challenges. One notable issue that has gained attention is its susceptibility to memory leaks in certain scenarios. While Streamlit excels in transforming data scripts into interactive web applications, it has been observed that under specific conditions, memory usage may steadily increase over time, ultimately leading to performance degradation and potential crashes. This phenomenon, known as memory leak, can be particularly concerning, especially in applications where stability and reliability are paramount.

Furthermore, Streamlit's compatibility with streaming large volumes of data, such as real-time video footage from devices like the Jetson Nano to a streaming server, has also raised concerns. The process of streaming data between devices demands efficient memory management and optimized network communication. However, in certain configurations, users had challenges in smoothly streaming data, with performance bottlenecks and interruptions impacting the overall user experience.

This led to both ImageZMQ and Streamlit being ruled out as potential tools for the final products.

7.4 MQTT

MQTT (Message Queuing Telemetry Transport) has emerged as a critical protocol in the realm of IoT and data communication. Designed for lightweight and efficient messaging, MQTT facilitates communication between devices and systems with minimal overhead, making it particularly suited for resource-constrained environments. The protocol's publish-subscribe architecture enables devices to exchange messages through a centralized broker, enhancing the scalability and flexibility of IoT applications.

One of the most widely used implementations of MQTT is the Mosquitto broker. Mosquitto provides an open-source and lightweight solution for setting up MQTT brokers, acting as a central hub that mediates the flow of messages between publishers and subscribers. The broker's efficient design and low memory footprint make it an ideal choice for diverse IoT scenarios, from home automation to industrial applications.

Mosquitto's configuration options allow users to tailor the broker to their specific requirements, such as authentication mechanisms, data retention policies, and access controls. This configurability not only ensures security but also enhances the efficiency of message routing and delivery. Moreover, Mosquitto supports various Quality of Service levels, providing users the flexibility to choose the desired level of message reliability and delivery guarantee.

MQTT and Mosquitto were introduced to solve an emerging problem during the project development: The Jetson Nano can stream (communicate) to the streaming server and by extension, the users, but the users can not communicate with the Jetson Nano to change its settings and output. The solution was to combine MQTT producers with the Streamlit interface to send messages that the Jetson Nano can receive and change its configurations accordingly. However, as the project grows intending to create an ecosystem for multiple users and Jetson Nano, which can communicate with each other, the MQTT solution was replaced by Apache Kafka, a distributed streaming platform that focuses on high-throughput, fault-tolerant, and scalable data streaming.

7.5 Apache Kafka

Figure 22: An open-source distributed event streaming platform - Apache Kafka

The landscape of data processing and streaming has witnessed a transformative shift with the emergence of Apache Kafka. As a distributed streaming platform, Kafka stands as a testament to the ever-growing demand for efficient real-time data pipelines and streaming applications. Born out of the innovation of LinkedIn and subsequently open-sourced by the Apache Software Foundation, Kafka has established itself as a cornerstone of modern data engineering ecosystems.

At its core, Kafka presents a revolutionary approach through its conceptualization of a distributed commit log. This approach revolutionizes traditional messaging systems by orchestrating a robust architecture capable of managing vast volumes of data streams while upholding exceptional throughput and fault tolerance. The driving force of Kafka's architecture is the Kafka broker, a versatile entity that seamlessly transitions between a message broker, a data storage system, and a dynamic data streaming platform. This pivotal piece forms the bedrock upon which Kafka's intricate ecosystem thrives.

Central to Kafka's ecosystem are the notions of topics and partitions. Topics act as conduits for data organization, carving logical channels through which data streams flow. Producers, in turn, are the architects of this flow, composing data records into Kafka topics. On the other end of the spectrum, consumers assume the role of data recipients, subscribing to topics and extracting value from the data records they hold.

Underpinning this structure is Kafka's innovative use of partitions. By dividing topics into partitions, Kafka optimizes data distribution and parallel processing. This segmentation equips

Kafka with the prowess to scale horizontally and efficiently manage voluminous data streams across a dynamic network of brokers.

Operationalizing Kafka necessitates a coherent orchestration of its key components. Producers harness the power of Kafka by infusing data records into specific topics, while consumers immerse themselves in the rich data stream by subscribing to these topics. Within this ecosystem, consumer groups emerge, embodying the collaborative nature of data processing. Kafka's partition-based consumption model ensures that each partition is processed by a single consumer within a group, thus harmonizing the load distribution and processing efficiency.

In practice, the journey of data within Kafka unfolds as a symphony of publishing, storage, partitioning, replication, and consumption. Data records are meticulously published by producers, encapsulating the essence of events within the digital realm. These records are subsequently etched into the storage of Kafka brokers, manifesting as messages within a commit log. This transformative storage approach fosters both scalability and durability, hallmarking Kafka's commitment to efficient and reliable data processing.

The tale of Kafka is one of evolution, as it not only complements but in this scenario supplants traditional protocols such as MQTT. While MQTT aptly serves lightweight communication needs, Kafka excels in data-intensive environments that necessitate intricate data processing and event-driven architectures. Kafka's dominance in scenarios requiring immense data volumes and complex data transformations is attributed to its robust publish-subscribe model, parallel processing capabilities, fault tolerance through data replication, and scalability through horizontal expansion.

Kafka worked so well as a lightweight distributed event-streaming platform that eventually, it was also implemented into the streaming architecture.

7.6 Video transmitting and streaming

The updated version of the project proposes implementing streaming to improve the automation of an AIoT system while enabling remote surveillance.

7.6.1 Overview

In earlier iterations, Streamlit played a pivotal role as the platform that facilitated real-time visualization of the AIoT process. Concurrently, ImageZMQ served as the application responsible for delivering frames, effectively serving as a conduit between ImageZMQ and Streamlit. This seamless synergy allowed Streamlit to efficiently broadcast the received frames to end users, enabling them to observe the AIoT operations as they transpired in real time.

Nevertheless, as elucidated in the preceding sections, the array of issues and drawbacks associated with these applications has reached a magnitude that renders the current system obsolete and impractical.

The replacement for the existing system entails a novel streaming approach inspired by prominent

platforms like YouTube and Twitch. To further enhance this setup, a Content Delivery Network (CDN) is integrated to proficiently regulate the distribution of content to users.

7.6.2 Theory and Testing

a) Video Streaming Basics

In the rapidly evolving landscape of AIoT, the traditional approach to video streaming has undergone a significant transformation, propelled by the principles observed in industry giants. This novel strategy revolves around a more sophisticated understanding of video streaming fundamentals, aimed at overcoming the limitations of the previous system and delivering an unparalleled user experience.

The foundation of modern video streaming rests upon the efficient collection and segmentation of video frames. This process takes inspiration from the best practices of platforms like YouTube and Twitch. Video frames, extracted from sources like cameras or edge devices' processing results, form the raw material for streaming content. Large quantities of seemingly insignificant frames can overwhelm. Video streaming inefficiency results from sending a constant large amount of data per second for a stable 30 frames per second video feed. This is where chunk compression comes into play. These frames are meticulously compressed into smaller, manageable segments known as chunks. This segmentation isn't merely an organizational tactic; it plays a pivotal role in ensuring smooth and adaptive video playback on various user devices.

b) Video Compression: .h264 and .h265

In the realm of video streaming, the art of compression is an essential factor that shapes the balance between data size and visual quality. Two prominent compression formats, .h264 (Advanced Video Coding) and .h265 (High-Efficiency Video Coding) have emerged as pivotal players in this dynamic landscape. Let's delve deeper into these formats, their characteristics, and the factors that influence their choice in modern video streaming architectures.

.h264, also known as AVC (Advanced Video Coding), has been a steadfast and reliable codec in the video compression domain. It holds a strong presence owing to its compatibility across a broad range of devices and its relatively efficient compression capabilities. .h264 achieves compression by utilizing predictive coding, wherein it encodes video frames by referencing previous and subsequent frames, identifying redundancies, and only transmitting the differences (motion vectors) between them. While it effectively reduces data size, .h264 strikes a balance between compression and quality, making it a suitable choice for applications where real-time streaming and moderate visual quality are desired.

As the demand for higher-quality content and more efficient compression has grown, .h265, also known as HEVC (High-Efficiency Video Coding), emerged as a game-changer. The key highlight of .h265 lies in its ability to deliver superior compression efficiency while preserving visual quality. This efficiency is achieved through advanced techniques such as larger block sizes,

enhanced motion vector prediction, and improved intra-frame prediction. By capturing more details and optimizing the encoding process, .h265 reduces data size significantly compared to its predecessor. However, it's important to note that the computational complexity required for encoding and decoding .h265 is higher than that of .h264, which can impact the hardware requirements for real-time streaming applications.

c) Optimizing Video Compression: An In-Depth Experiment

In the quest for optimizing video compression, a meticulous experiment has been designed to explore the most efficient compression type under varying frame sizes. Furthermore, the experiment incorporates an investigation into the impact of the chunk length, defined by the number of frames compressed within each segment. The objective of this test is to shed light on the optimal compression strategy, providing insights that can significantly influence video streaming systems' performance.

The size of a video frame profoundly influences its storage requirements, transmission bandwidth, and overall data volume associated with the image or video. To capture this effect, a range of frame sizes has been selected. For each frame size, a set of frames is established and consistently employed across all tests. This ensures that the impact of sizes on compression can be accurately measured. Afterward, to quantify the data requirements for each frame size, the experiment computes the Mean Size of Frames within each dimension category. This metric provides a clear understanding of the average data volume associated with different frame sizes. As mentioned, .h264 and .h265 are two Compression Formats being evaluated in the experiment. Both formats are widely used in video compression, but their performance characteristics can differ significantly. Comparing these formats under different conditions allows for an informed choice of the most suitable codec for specific applications.

After compressing frames into chunks using the defined settings, the experiment captures the range of resulting Video Sizes. This data is essential for assessing the overall size of compressed videos for each combination of variables. The mean Video Size for each chunk configuration is calculated. To gain insights into how compression affects individual frames within a chunk, the experiment computes the Size per Frame. This metric is obtained by dividing the mean video size for a chunk by the number of frames it contains. It provides a granular view of data utilization within a chunk.

Finally, to gauge the efficiency of each compression setting, the experiment calculates the efficiency metric. This is achieved by dividing the mean frame size by the size per frame within a chunk. A higher efficiency value indicates that the compression setting optimally utilizes the available data.

Following an extensive testing phase, here are the conclusive results.

Testing Context							Results					
W x H	FS	Min	Max	Mean	NoF	CF	VS	Min	Max	Mean	SpF	Eff
640 x 480	61.3-80.4	61.3	80.4	70.85	5	.h264	17.7-144	17.7	144	80.85	16.17	4.38
					10	.h264	53.7-88	53.7	88	70.85	7.09	9.99
					20	.h264	128.4-280	128.4	280	204.2	10.21	6.94
					30	.h264	169-377	169	377	273	9.1	7.79
					40	.h264	260-391	260	391	325.5	8.14	8.7
					50	.h264	319.9-556	319.9	556	437.95	8.76	8.09
					60	.h264	316.3-595.9	316.3	595.9	456.1	7.6	9.32
					70	.h264	448.1-629.2	448.1	629.2	538.65	7.7	9.2
					80	.h264	683.1-962.9	683.1	962.9	823	10.29	6.89
					90	.h264	504-1000	504	1000	752	8.36	8.47
					100	.h264	595-1100	595	1100	847.5	8.48	8.35
					5	.h265	8.05-47.8	8.05	47.8	27.93	5.59	12.67
					10	.h265	10.9-81.6	10.9	81.6	46.25	4.63	15.3
					20	.h265	14.9-150	14.9	150	82.45	4.12	17.2
					30	.h265	19.9-218	19.9	218	118.95	3.97	17.85
					40	.h265	22.8-277	22.8	277	149.9	3.75	18.89
					50	.h265	26.3-340	26.3	340	183.15	3.66	19.36
					60	.h265	28.3-389	28.3	389	208.65	3.48	20.36
					70	.h265	30.7-405	30.7	405	217.85	3.11	22.78
					80	.h265	33.6-464	33.6	464	248.8	3.11	22.78
					90	.h265	39-511	39	511	275	3.06	23.15
					100	.h265	41.6-520	41.6	520	280.8	2.81	25.21
1280 x720	208-211	208	211	209.5	5	.h264	109-203	109	203	156	31.2	6.71
					10	.h264	153.2-317.7	153.2	317.7	235.45	23.55	8.9
					20	.h264	321-619	321	619	470	23.5	8.91
					30	.h264	552.3-813	552.3	813	682.65	22.76	9.2
					40	.h264	616.5-1000	616.5	1000	808.25	20.21	10.37
					50	.h264	855.2-1200	855.2	1200	1027.6	20.55	10.19
					60	.h264	1100-1400	1100	1400	1250	20.83	10.06
					70	.h264	1200-1700	1200	1700	1450	20.71	10.12
					80	.h264	1200-1900	1200	1900	1550	19.38	10.81
					90	.h264	1300-2300	1300	2300	1800	20	10.48
					100	.h264	1500-2600	1500	2600	2050	20.5	10.22
					5	.h265	54.3-80.9	54.3	80.9	67.6	13.52	15.5
					10	.h265	98.9-116	98.9	116	107.45	10.75	19.49
					20	.h265	112-183	112	183	147.5	7.38	28.39
					30	.h265	160-245	160	245	202.5	6.75	31.04
					40	.h265	198-334	198	334	266	6.65	31.5
					50	.h265	194-386	194	386	290	5.8	36.12
					60	.h265	232-445	232	445	338.5	5.64	37.15
					70	.h265	298-492	298	492	395	5.64	37.15
					80	.h265	310-588	310	588	449	5.61	37.34
					90	.h265	348-597	348	597	472.5	5.25	39.9
					100	.h265	488-662	488	662	575	5.75	36.43
					5	.h264	207-2090	207	2090	1148.5	229.7	7.84
					10	.h264	2100-3600	2100	3600	2850	285	6.32

2560 x 1440	1700-1900	1700	1900	1800	20	.h264	4400-6800	4400	6800	5600	280	6.43
					30	.h264	7400-10700	7400	10700	9050	301.67	5.97
					40	.h264	11200-12300	11200	12300	11750	293.75	6.13
					50	.h264	12700-17200	12700	17200	14950	299	6.02
					60	.h264	17600-23200	17600	23200	20400	340	5.29
					70	.h264	23500-28000	23500	28000	25750	367.86	4.89
					80	.h264	29100-30600	29100	30600	29850	373.13	4.82
					90	.h264	31200-36000	31200	36000	33600	373.33	4.82
					100	.h264	36900-38400	36900	38400	37650	376.5	4.78
					5	.h265	516-997	516	997	756.5	151.3	11.9
					10	.h265	648-1250	648	1250	949	94.9	18.97
					20	.h265	1350-2010	1350	2010	1680	84	21.43
					30	.h265	2010-2720	2010	2720	2365	78.83	22.83
					40	.h265	2740-3470	2740	3470	3105	77.63	23.19
					50	.h265	3410-4290	3410	4290	3850	77	23.38
					60	.h265	4200-4880	4200	4880	4540	75.67	23.79
					70	.h265	4580-5750	4580	5750	5165	73.79	24.39
					80	.h265	5290-6340	5290	6340	5815	72.69	24.76
					90	.h265	6120-7030	6120	7030	6575	73.06	24.64
					100	.h265	6770-7950	6770	7950	7360	73.6	24.46

Table 7: Compression type effectiveness across different video frame sizes

W x H = *Width X Height (pixel)*, **FS** = *Frame Size (KiB)*, **NoF** = *Number of Frames*, **CF** = *Compression Formats*, **VS** = *Video Size (KiB)*, **SpF** = *Size per Frame (KiB)*, **Eff** = *Efficiency*

d) Experiment Results

The results of the experimentation regarding the .h264 and .h265 compression formats about various frame sizes provide valuable insights. Both compression formats exhibited a similar pattern: as frame sizes increased, the efficiency of compression also increased. This suggests that larger frames, which inherently contain more visual detail, benefit from more efficient compression. Notably, both .h264 and .h265 experience a decline observed at the largest frame size setting, with .h264 having a harsher fall. This decline in efficiency might be attributed to the limitations of the .h264 codec when handling extremely large frames.

In contrast, the .h265 compression format demonstrated superior efficiency, as anticipated. .h265 is well-known for its ability to efficiently compress high-resolution content. However, it's important to note that .h265 compression typically demands more advanced hardware resources for encoding and decoding due to its complexity.

Despite the advantages of .h265 compression, it appears that the Jetson Nano, the hardware used in these tests, faced challenges in efficiently handling .h265 compression. The time required for the Jetson Nano to compress .h265 video and subsequently transmit and stream it seems to exceed acceptable limits. This could result in noticeable downtime or delays in the streaming process.

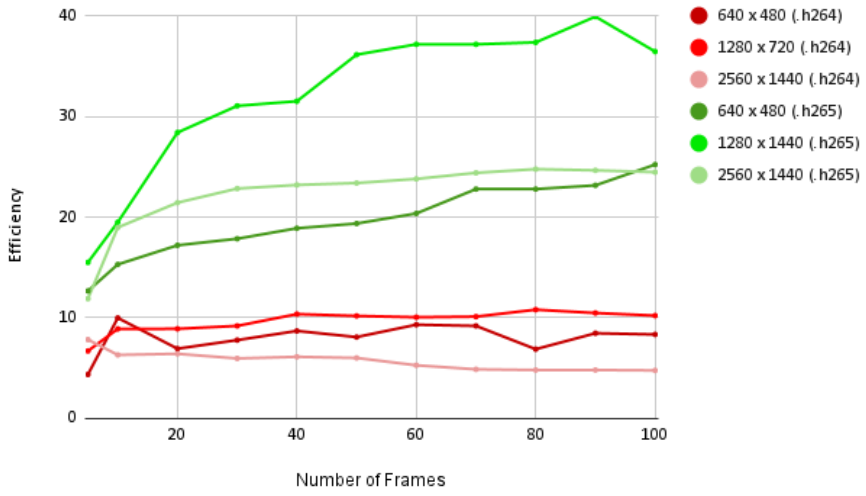


Figure 23: Comparing between efficiency of different Video Compressions

7.6.3 Content Delivery Network

A Content Delivery Network emerges as a central player in modern video streaming architectures. This network of strategically placed servers optimizes the delivery of content to end-users. By selecting the server closest to a user's geographical location, a CDN minimizes latency and accelerates content retrieval. This not only enhances the user experience but also ensures scalability and load balancing across the network, reducing the burden on the primary streaming server.

As alluded to previously, Apache Kafka will not only enable users to communicate with the edge device but also serve as a video chunk delivery system from the Jetson Nano to the CDN. The compressed video chunks are efficiently transported using the Apache Kafka messaging system. Apache Kafka excels in managing data streams, offering high throughput and fault tolerance. It ensures that the compressed chunks are reliably delivered to the intended recipients, minimizing data loss and ensuring seamless playback.

The compressing and streaming of the video chunks are handled by FFmpeg. FFmpeg excels in addressing the intricate challenges of multimedia compression. It is the go-to solution for converting collections of images, or frames, into compressed videos. This process is crucial for reducing file sizes while preserving visual quality, making it ideal for applications like video storage and transmission.

In the context of streaming, FFmpeg shines as an adaptive bitrate streaming solution. This technology ensures that viewers receive the best possible quality based on their internet connection. FFmpeg dynamically adjusts the video quality by encoding video content at multiple bitrates and resolutions, allowing seamless transitions between different qualities as network conditions fluctuate. This ensures a smooth streaming experience, preventing buffering issues and providing a superior viewing experience for a global audience.

Beyond video compression and adaptive streaming, FFmpeg's streaming capabilities are unparalleled. It supports various streaming protocols, including Real-Time Messaging Protocol (RTMP) and HTTP Live Streaming (HLS). These protocols enable the efficient delivery of video content to a wide range of devices and platforms, making FFmpeg a go-to choice for broadcasters, content creators, and live streamers.

Within the CDN architecture, a specialized receiver is configured to capture incoming Kafka messages seamlessly. These messages encapsulate the compressed video chunks, ready for further processing and distribution. Subsequently, the CDN orchestrates the efficient streaming of these video chunks to the dedicated Streaming Server. This harmonious flow of data ensures the smooth and uninterrupted journey of video content from its source to the awaiting audience.

7.6.4 Streaming Server and Adaptive Streaming

Figure 24: An open-source robust web server - NGINX

The Streaming Server is set up using NGINX, a robust and versatile web server. Its efficient and scalable architecture makes it an ideal choice for delivering multimedia content over the Internet. This dynamic duo orchestrates a seamless and efficient video streaming experience.

NGINX serves as the cornerstone of the Streaming Server setup. Its primary responsibility is to act as the endpoint for the incoming video chunks, diligently received from the CDN. These video chunks, previously compressed by FFmpeg, are handed over to NGINX for further processing and dissemination. Here, NGINX ensures that the video stream is properly structured, authenticated, and made available to users.

NGINX is set to stream the video footage to an RTMP channel. While the traditional RTMP serves as a standard method for delivering streams, its compatibility with modern browsers is limited. To overcome this challenge, streams are further adapted using the HLS protocol. This protocol divides the video into smaller segments and dynamically adjusts the quality based on the user's network conditions.

8 Proposed systems

With all the prerequisites established, this section is concerned with the architecture of the system with its evolution throughout different versions.

8.1 Version 0.1

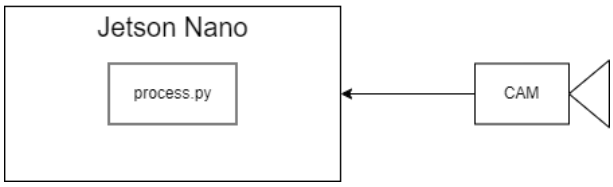


Figure 25: System Workflow V0.1a

At the core of the system is the implementation of YOLOv8 to work with the Jetson Nano architecture. Adapting YOLOv8 to the Jetson Nano involves optimizing the model to make the best use of the device's CPU and GPU resources. This ensures that real-time object detection remains feasible without overwhelming the hardware.

YOLOv8 will take a camera feed as an input for detection. It does not have to be a physical camera connecting the Jetson Nano, as other streaming sources like RTSP also work. This flexibility allows the system to seamlessly integrate with a wide range of cameras and streaming devices, making it adaptable to different surveillance and monitoring setups.

With YOLOv8 running real-time detections, a script is built using the fundamentals of YOLO and Supervision as mentioned in previous parts to log the important information. Even so, this is just the Artificial Intelligence part of the AIoT. To call this an AIoT system, an IoT database has to be introduced, as seen in Figure 26.

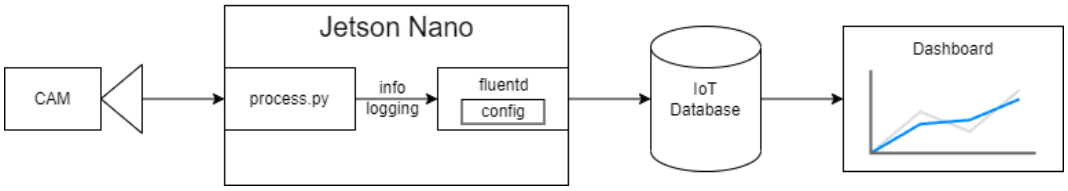


Figure 26: System Workflow V0.1b

This marks the first official version of the product, representing a significant milestone in its development. It introduces Fluentd as a crucial component of the system, responsible for monitoring and transmitting the logged information to the designated IoT database. Once the data reaches the IoT database, it undergoes a comprehensive analysis, compilation, and presentation process.

One of the standout features of this version is the ability to present the analyzed data to users through a variety of intuitive graphical elements. These include line graphs that provide visual

insights into trends and patterns, as well as raw tables for detailed data exploration. Additionally, the system incorporates interactive elements such as buttons, switches, and knobs, each with customizable conditions for execution.

8.2 Version 0.2

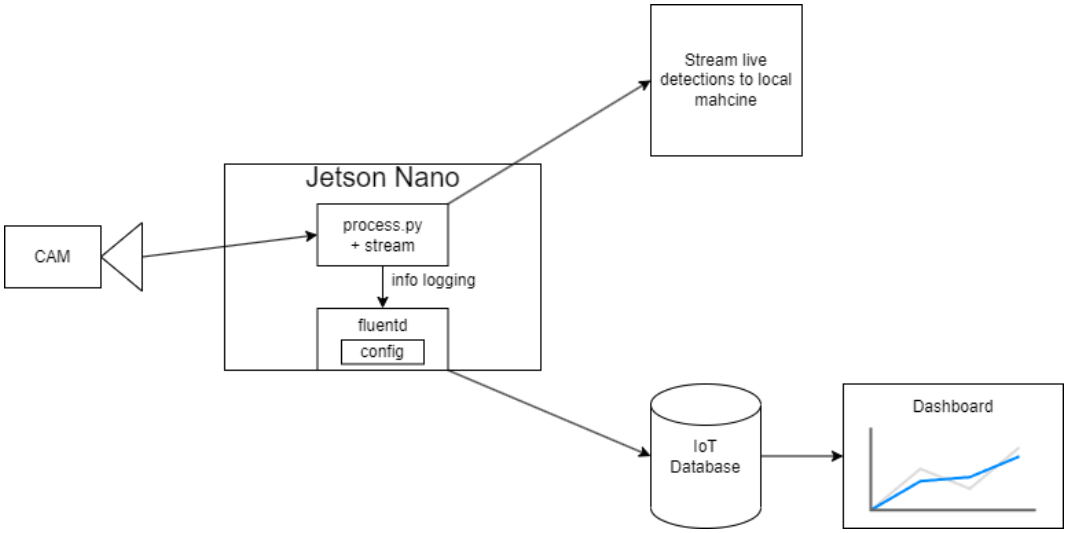


Figure 27: System Workflow V0.2a

Starting from this version, the project undergoes a strategic shift in focus, prioritizing the efficient delivery of data to users. While the core functionality remains centered around AIoT, the emphasis now lies on enabling users to access live detections and offering them the ability to interact with YOLO's settings on the Jetson Nano. This evolution led to the introduction of a surveillance and monitoring system in V0.2a.

In this particular version, a key modification involves altering the Python script's processing logic. The script is now configured to stream every post-processed frame to a local machine. This adjustment marks a significant enhancement in the system's functionality. By streaming each frame to a local device, users gain real-time access to the processed frames, enabling them to monitor and analyze the data as it is generated.

Nevertheless, this adjustment introduced a limitation: only local devices could access and view the processing. While this is acceptable for specific use cases, the aim is to create a versatile system that accommodates a wide range of scenarios. Ideally, users should have the flexibility to customize or scale back access as needed to suit their requirements.

To address this, the subsequent version of the system introduces streaming to a public IP server, as seen in Figure 28.

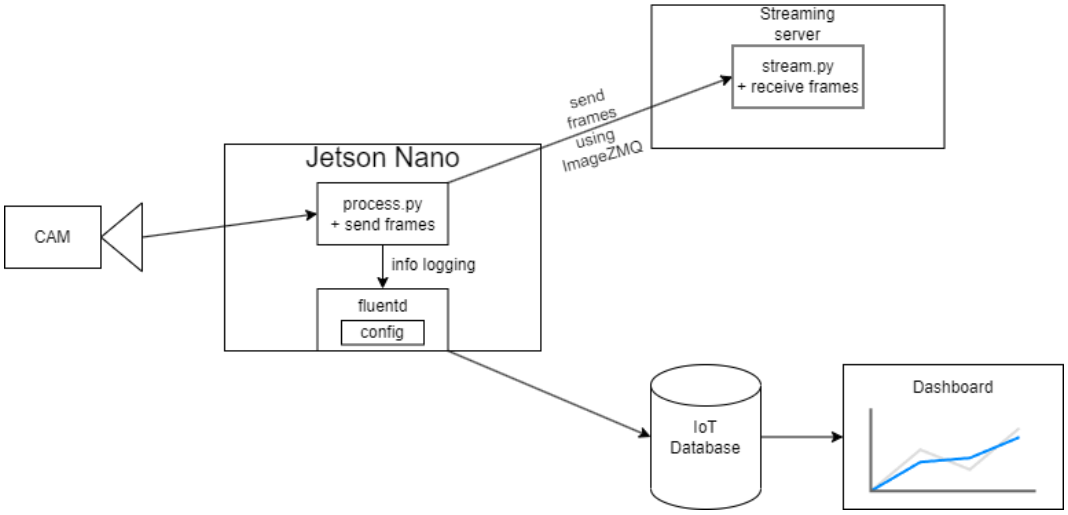


Figure 28: System Workflow V0.2b

As elaborated in the previous sections, this particular version of the project utilized ImageZMQ for transmitting frames to the Streaming Server. Subsequently, these frames were streamed through the server's public IP address. In doing so, the users can access the server from the public IP to see the process in real-time.

However, a significant issue with this version is the strain it places on the Jetson Nano. Not only does it have to process each frame, but it must also immediately send that frame to the Streaming Server. This approach can lead to problems, especially in scenarios where the edge device loses its internet connection. In such cases, the entire process stalls, and incoming frames are not processed.

This limitation poses a significant challenge, as real-time processing and seamless data transmission are critical for the system's effectiveness. It also poses major security issues. As such, the version illustrated in Figure 29 breaks the scripts into more manageable segments and introduces Streamlit.

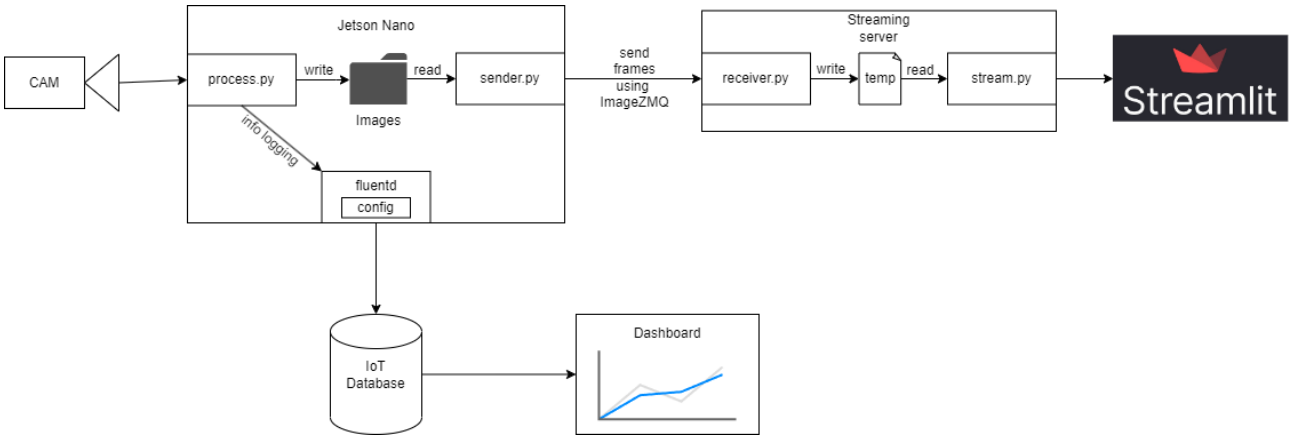


Figure 29: System Workflow V0.2c

In this version, two new Python scripts, namely "sender" and "receiver," have been introduced, separating them from their original scripts. To enable communication between the process and stream components and the sender and receiver scripts, temporary folders have been created. These folders are used to store processed frames and received frames, respectively.

This architectural improvement empowers the Jetson Nano to maintain its frame processing capabilities even when it experiences interruptions in internet connectivity. The process script operates autonomously, detecting, logging information, and storing frames in dedicated folders during periods when internet access is unavailable. Once the internet connection is reestablished, Fluentd and ImageZMQ can resume their operations.

To ensure that users receive real-time footage and are not subjected to delayed frame delivery, a subsystem has been integrated. When the number of frames in the folder surpasses a predefined threshold, the system automatically initiates frame management. Specifically, it deletes a designated number of frames to prioritize the delivery of the most up-to-date footage to users. This mechanism guarantees a seamless viewing experience for users, even in scenarios with temporary internet disruptions.

This architectural change enhances the modularity and flexibility of the system. By isolating the sender and receiver functionalities into separate scripts and using temporary folders as intermediaries, the system gains more control and adaptability in managing data transmission. Additionally, this approach helps streamline the data flow and ensures that frames are efficiently processed and transmitted between the different components of the system.

In the project's evolution, up to this stage, the Jetson Nano effectively communicates with the Streaming Server, facilitating the distribution of data to users. However, this communication remains unidirectional, allowing data to flow from the Jetson Nano to the Streaming Server and subsequently to users. To establish a bidirectional communication channel, MQTT, utilizing Mosquitto as the broker, is introduced in the project's next version. This pivotal addition enhances the system's capabilities, enabling both data transmission and reception between the Jetson Nano and the Streaming Server, thus fostering more interactive and dynamic user experiences.

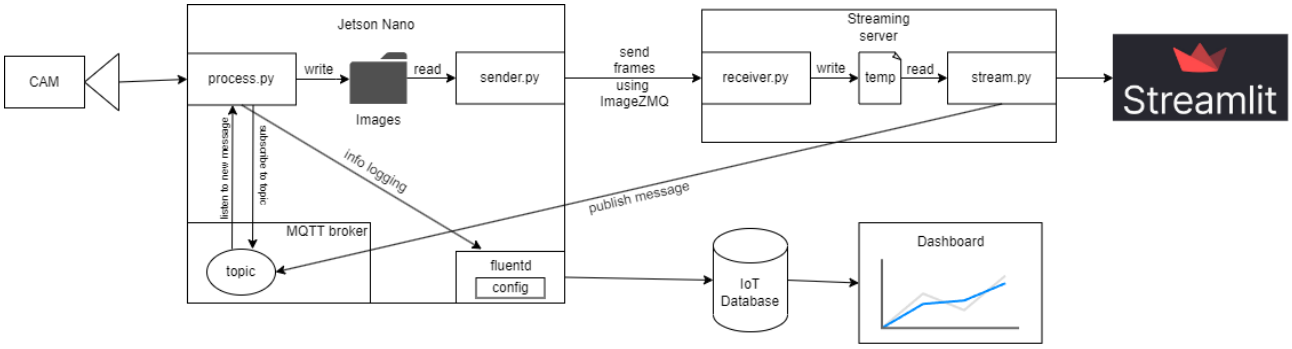


Figure 30: System Workflow V0.2d

In this version, which serves as a bridge to a major architectural transition, the project maintains a workflow similar to its predecessors. The notable addition is the implementation of an MQTT broker. A Streamlit script for streaming video is crafted to provide users with enhanced control over the system. Users can now manipulate settings such as confidence levels, IOU thresholds, and select from a range of YOLO models, spanning from YOLOv5 to YOLOv8.

These user-selected options trigger messages sent to a specific topic on the MQTT broker. The process script continuously monitors this topic, awaiting the latest messages dispatched by the MQTT publisher. Upon receipt, these messages are translated into new settings for the Jetson Nano. Users may experience a slight delay before observing the effects of their requested changes in their video feeds. This setup establishes a dynamic and responsive interaction between users and the AIoT system, enhancing the overall user experience.

At this juncture, the project encountered a pivotal challenge: scalability. The existing components, namely ImageZMQ and the MQTT broker facilitated by Mosquitto, began to reveal their limitations in accommodating the project's expanding needs.

ImageZMQ, while proficient at its task of transporting image frames from the Jetson Nano to the streaming server, exhibited constraints when dealing with a growing number of edge devices. Each device is expected to send 3 streams of images at once to accommodate the users' needs. As more edge devices were added to the system, the load on ImageZMQ increased significantly. This surge in demand strained the capabilities of ImageZMQ, causing delays and potential bottlenecks in the frame delivery process.

Similarly, the MQTT broker, although effective for managing communication between the edge devices and the Jetson Nano, demonstrated limitations in handling an extensive network of devices. As the number of devices grew, the MQTT broker faced challenges in efficiently routing messages, potentially resulting in congestion and delays in message delivery. This hindered the seamless coordination of settings and commands between the users and the edge devices.

Concurrently, Streamlit, which had served adequately for visualizing and presenting complex graphs, began to demonstrate shortcomings when tasked with the additional responsibilities of receiving and streaming frames. This expanded functionality placed a considerable burden on Streamlit, resulting in memory leaks and frequent crashes within the system.

8.3 Version 0.3

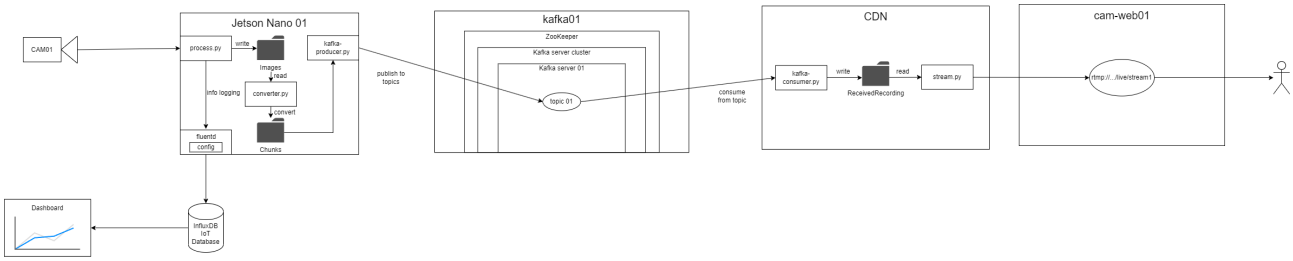


Figure 31: System Workflow V0.3a

This version represents a significant departure from the project's earlier iterations, marking a substantial shift in its architectural design. Here, Apache Kafka assumes a central role, superseding its predecessors in facilitating communication between users and edge devices. Additionally, the streaming protocol undergoes a substantial transformation, replacing ImageZMQ with Apache Kafka as its core communication component.

The adoption of Apache Kafka in this version is pivotal, as it introduces a more robust and scalable communication infrastructure. Kafka's distributed and fault-tolerant architecture enhances the system's capacity to handle large-scale data flows, ensuring the seamless transmission of information between edge devices and users. This transition signifies a critical step towards enhancing the project's scalability and performance, addressing previous limitations associated with ImageZMQ and MQTT.

Furthermore, the revamped streaming protocol, now reliant on Apache Kafka, marks a fundamental alteration in how video frames are managed and transmitted. This adaptation aims to leverage Kafka's capabilities in managing real-time data streams efficiently. By integrating Apache Kafka into the streaming process, the project endeavors to provide a more reliable and responsive video streaming experience, catering to the diverse needs of its users.

While Apache Kafka brings robustness to the system, the continuous transmission of raw frames at a consistent rate of 30 frames per second per stream imposes considerable strain on the Kafka infrastructure. To address this challenge and optimize data transmission, a novel approach is introduced: the consolidation of frames into video chunks. This innovative strategy involves compressing multiple frames into a single video chunk, offering significant bandwidth savings and alleviating the burden on the Kafka ecosystem.

Additionally, the system incorporates a Streaming Server that is configured through NGINX to receive the streamed content originating from a CDN. This architectural element plays a pivotal role in the real-time content distribution process, ensuring that data is efficiently routed and delivered to end-users.

The CDN streams content using FFmpeg to an RTMP stream hosted by NGINX. This stream signal can be received using applications such as Open Broadcaster Software (OBS) or VideoLAN Client (VLC). However, modern browsers have withdrawn their support for RTMP streams, so an additional step needs to be taken.

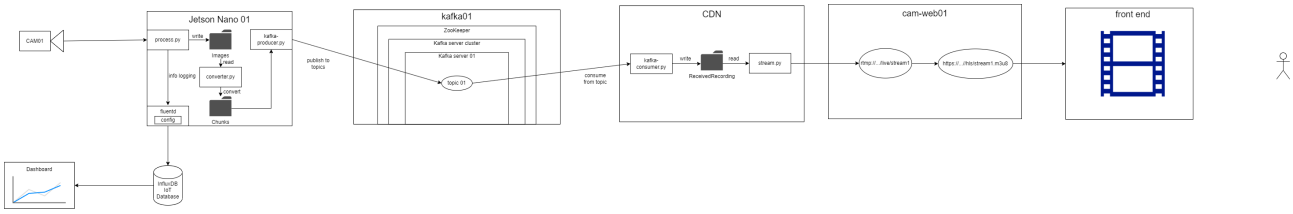


Figure 32: System Workflow V0.3b

Version 0.3b introduces significant enhancements to the NGINX configuration, aimed at further optimizing the project's streaming capabilities. One of the key improvements is the integration of HLS.

HTTP Live Streaming is a widely adopted streaming protocol developed by Apple. It breaks multimedia content into small chunks and uses a manifest file (M3U8) to describe the available streams and their quality levels. This addition allows for adaptive streaming, where the quality of the stream can adapt in real time based on the viewer's network conditions, ensuring a smooth viewing experience.

Unlike RTMP, HLS is compatible with a wide range of devices and platforms, including iOS and Android devices, web browsers, and smart TVs. This broad compatibility ensures that your content can reach a diverse audience.

This latest iteration of the project marks a significant milestone in its ongoing development. Currently, a front-end web application is being developed, representing a forward-looking approach to enhancing the user experience and expanding the project's capabilities.

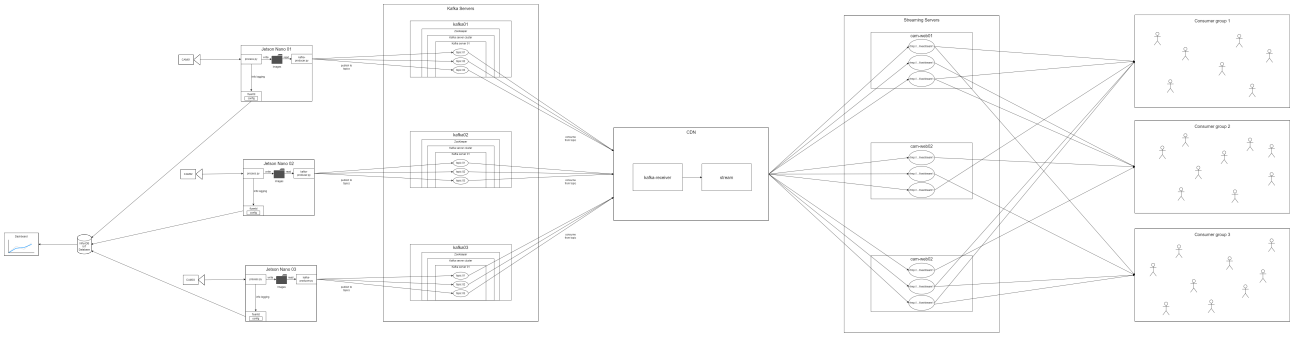


Figure 33: System Workflow V0.3 idea

The aim is to enhance the system's scalability to accommodate a growing user base and expanding network of edge devices. This involves ensuring that the infrastructure can seamlessly scale to meet the rising demands for streaming content. Additionally, the introduction of a front-end application allows for user credentialing and role assignment, reinforcing security and granting access to specific edge devices as needed.

9 Extension: Facial Recognition

While not initially within the scope of the original project, the system's adaptability allows for flexible modifications to cater to various requirements and evolving needs. An example of this is to incorporate facial recognition into it.

9.1 Extension Context

The proposal to implement facial recognition for student attendance at VGU comes as a response to the institution's growth and the need for a more efficient and secure attendance-tracking system. As the student population continues to expand, the traditional methods of taking attendance have become increasingly time-consuming and susceptible to fraudulent practices. This extension aims to leverage facial recognition technology to address these challenges and streamline the attendance-checking process.

VGU, like many educational institutions, faces challenges associated with manually taking attendance. These challenges include the need for instructors to individually mark each student's presence, which can be time-consuming, especially in larger classes. Or the method of passing attendance papers can be cheated by having a friend sign them for you.

The introduction of facial recognition technology for attendance tracking, coupled with remote monitoring capabilities, has the potential to revolutionize traditional attendance management systems.

9.2 FaceNet

9.2.1 Overview

Face recognition technology has made remarkable strides in recent years, with FaceNet standing as a pinnacle achievement in the field. Developed by researchers at Google, FaceNet is an innovative deep-learning model designed for face recognition and verification tasks. It's renowned for its ability to create highly discriminative facial embeddings, allowing it to accurately identify individuals in images and videos. This comprehensive overview delves into the workings and significance of FaceNet in the realm of computer vision.

FaceNet's groundbreaking achievement lies in its ability to transform facial images into a compact yet highly informative vector representation known as an "embedding." These 128-dimensional vectors are the essence of a person's face, capturing the most critical facial features in a mathematical form.

But why use the term "embedding"? The concept is borrowed from mathematics and machine learning. An embedding is essentially a mapping that converts complex, high-dimensional data, such as an image, into a simplified, lower-dimensional representation while retaining its essential characteristics. In the case of FaceNet, it condenses a face image into a 128-number vector. This transformation is akin to distilling the essence of a face into a concise set of numbers.

9.2.2 Recognising Process

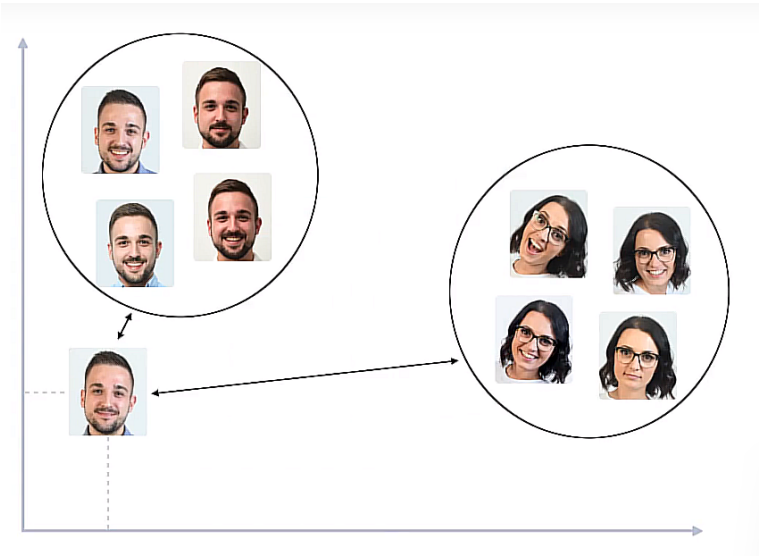


Figure 34: FaceNet images plotted in a 2D plain

FaceNet offers an efficient and effective means of recognizing individuals in unseen images. This recognition process primarily relies on calculating embeddings, computing distances, and making informed decisions based on these key components.

When presented with an unseen face in an image, FaceNet first calculates an embedding for this face. This embedding is a vector of 128 numbers, capturing the essential facial features.

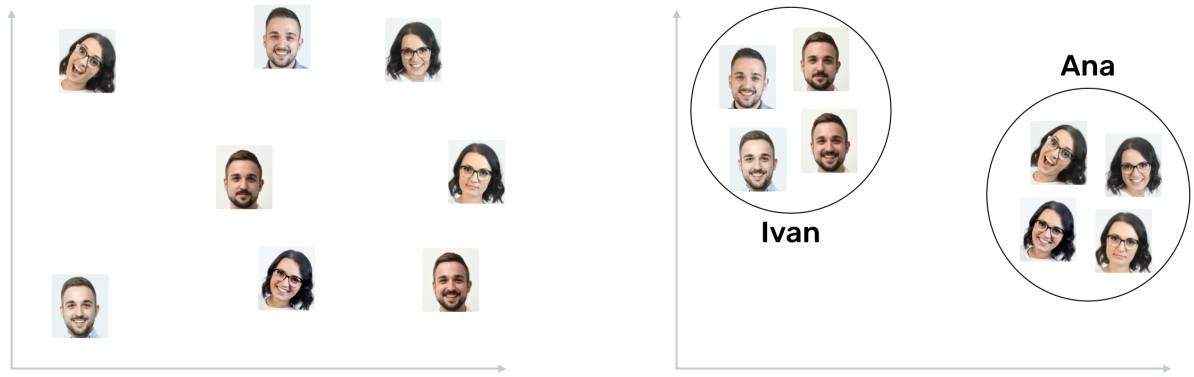
The system proceeds to compute the distances between the embedding of the unseen face and the embeddings of known individuals. This involves calculating the similarity (or dissimilarity) between the vectors.

The recognition outcome is based on the closest match (i.e., the reference embedding with the smallest distance). If multiple known individuals meet the threshold, FaceNet can return multiple potential matches or apply additional criteria to refine the decision.

9.2.3 Training Process

As for any facial recognition module, it needs to be trained and re-trained to add new faces to be identified. The process of doing so can boiled down to a few simple steps.

FaceNet begins by randomly selecting an "anchor image" from the dataset. This serves as a reference point in the learning process. To train FaceNet effectively, it then randomly selects another image featuring the same person as the anchor image. This image forms what's known as a "positive example." In parallel, FaceNet randomly picks an image featuring a different person from the anchor image. This image is termed a "negative example." Then, FaceNet's neural network is fine-tuned iteratively to optimize the embeddings' quality. The objective is to ensure that positive examples are closer to the anchor image in the embedding space than negative



(a) Init state before training

(b) Final state after training

Figure 35: FaceNet Training Process

examples. The network parameters, the internal settings that govern how the neural network operates, are adjusted during each iteration. This adjustment is guided by a loss function that quantifies the similarity between the anchor, positive, and negative embeddings.

These steps are repeated iteratively, ensuring that changes continue until a specific criterion is met. This criterion signifies that the embeddings of faces from the same person are indeed close to each other while being distinctly separated from those of different individuals.

9.3 YOLO + FaceNet

9.3.1 Overview

In the context of FaceNet, the process of face recognition involves two critical components: face detection and facial feature extraction using embeddings. While FaceNet excels at the latter, it relies on a robust face detection system to identify and locate faces within images during both the training and recognition phases. The YOLO algorithm plays a pivotal role in this process.

9.3.2 Detecting Faces

Leveraging the YOLOv8 algorithm, when specifically trained to focus solely on facial detection, it collaborates seamlessly with FaceNet to facilitate accurate facial recognition.

YOLOv8 training for facial detection represents a strategic approach to optimizing the performance of facial recognition systems. By narrowing the model's focus to the detection of individual faces, it helps achieve superior precision, efficiency, and adaptability across a range of applications, ultimately leading to more reliable and accurate facial recognition outcomes.

(a) World's Largest Selfie

(b) yolov8-face.pt

Figure 36: Running Face Detection on the World's Largest Selfie

9.3.3 Usage in Training Phase

The training phase for face recognition involves a series of steps designed to prepare the system for accurate and reliable identification of individuals. A vital component of this phase is the integration of YOLO for face detection and a face alignment process.

The first step involves taking images of the individuals who need to be identified and organized into folders, each named after the respective person's name. This step creates a structured dataset for training. Afterward, YOLO is employed as the face detection tool. YOLO's primary task is to analyze each image and identify the presence of faces. When a look is detected, YOLO records the coordinates of the bounding box around the face.

Once YOLO has identified and located faces within the images, the next step involves face alignment. Aligned faces are essentially images where the face is oriented consistently, making it easier for subsequent recognition steps. YOLO saves the aligned faces separately in a designated folder structure. This new folder structure maintains a similar organization to the original dataset, with aligned faces grouped by the individual's name.

Finally, FaceNet takes over the training phase to create a new classifier, which includes all faces in the aligned faces folder, using the mentioned training process.

9.3.4 Usage in Recognition Phase

In recognition mode, the collaborative efforts of YOLO and FaceNet continue to shine, working seamlessly to identify and verify faces within a given frame. YOLO takes the lead when a frame or image is presented for face recognition. It scans the entire image to identify the presence of faces. YOLO's object detection capabilities excel in accurately locating objects within the frame. YOLO not only identifies the presence of faces but also precisely outlines and locates the boundaries of each detected face within the frame. This is critical for pinpointing the specific face of interest, especially in scenarios where multiple faces may be present.

Once YOLO successfully detects and locates the relevant face within the frame, it passes this cropped face region to FaceNet. YOLO essentially serves as the "scout" that identifies and

prepares the face for recognition. With the cropped face region in hand, FaceNet goes to work. It calculates an embedding for the detected face, creating a unique numerical representation of its features. This embedding is then compared to the embeddings of known individuals stored in a database. If the calculated embedding for the detected face closely matches one of the stored embeddings, FaceNet identifies the individual associated with that matching embedding. This process results in the recognition of the person within the frame.

The YOLO-FaceNet integration allows for real-time face recognition. As frames or images are processed, YOLO identifies and prepares the faces, and FaceNet verifies their identities swiftly and accurately.

9.4 Implementation

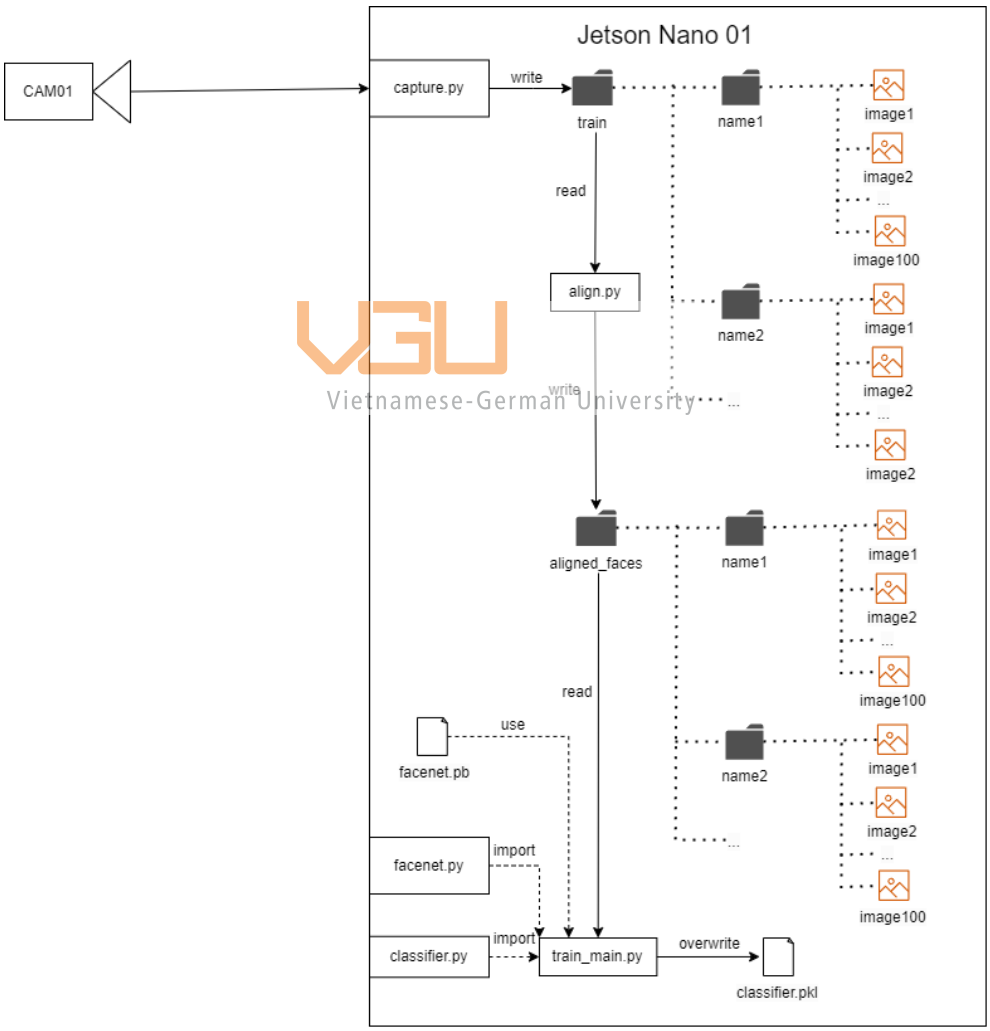


Figure 37: Training Phase Implementation

This Python script serves the purpose of capturing multiple images in bulk, to collect 100 photos for each individual. The process begins by prompting the user to input the name of the individual whose images are to be captured. Subsequently, the captured photos are stored within a folder named after the respective individual. Additionally, these individual folders are housed within

a parent directory labeled 'train'. The resulting directory structure closely resembles the visual representation depicted in Figure 37.

Following the bulk image capture process, the next step involves the utilization of an alignment script. This script leverages the capabilities of the YOLOv8 face detection model to precisely crop and store the detected faces in a designated folder termed 'aligned_faces'. The primary objective of this script is to identify and isolate the facial region that is most centrally positioned within a given photograph. This approach effectively eliminates the potential scenario of multiple faces being detected in a single image, which could otherwise introduce complexity and confusion into the dataset. By focusing on the most prominent facial feature in each image, the alignment script ensures that the dataset remains clean, coherent, and well-suited for subsequent tasks such as facial recognition or analysis.

In the last phase of the process, the next critical step is to train the dataset that has been meticulously curated. This training process is accomplished by harnessing a pre-trained FaceNet module, coupled with the utilization of 'facenet.py' and 'classifier.py'. These tools work cohesively to generate a 'classifier.pkl' file, which holds paramount significance in the subsequent facial recognition operations.

The 'classifier.pkl' file essentially serves as the cornerstone of the facial recognition process. Whenever a new face is introduced into the recognition system, this file is responsible for ensuring its accurate identification. To facilitate this seamless integration, the 'classifier.pkl' file must overwrite any outdated versions. However, there are specific scenarios, such as when attendance checks are carried out for particular classes or groups, where the current 'classifier.pkl' file may not need to be modified or overwritten. In these cases, a modification of the code is needed.

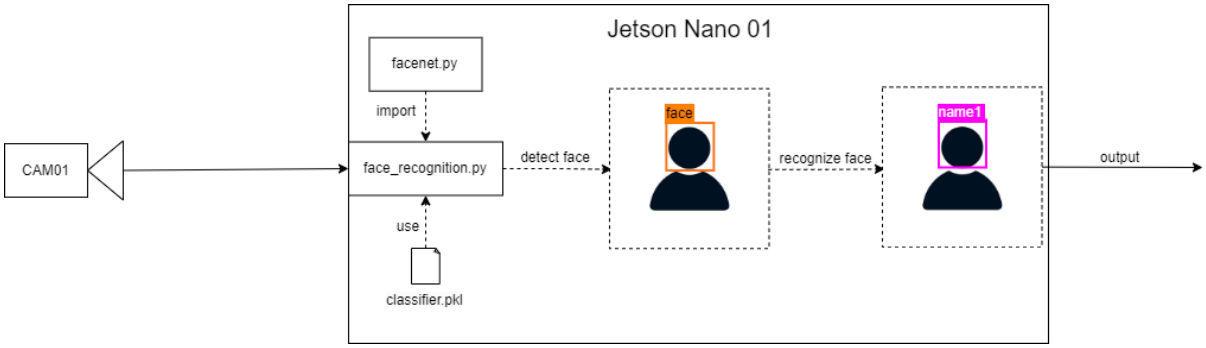
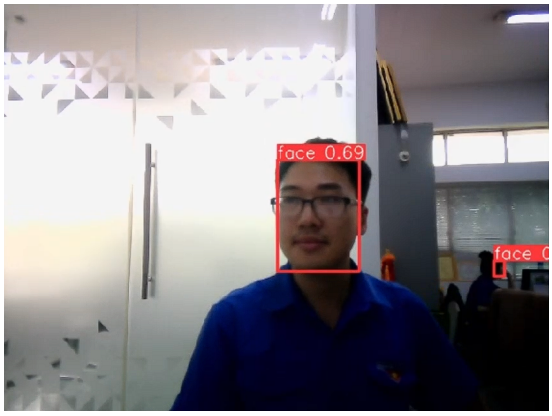


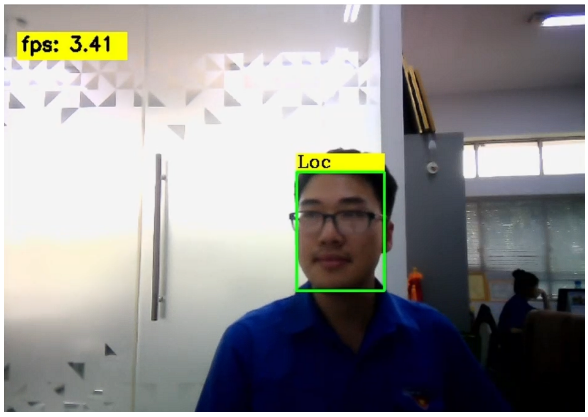
Figure 38: Recognition Phase Implementation

During the Recognition Phase, a Python script optimized for facial recognition is employed, utilizing the latest iteration of the 'classifier.pkl' file in conjunction with the 'facenet.py' script. The primary objective at this stage is to locate faces within a single frame extracted from the camera feed. YOLOv8 plays a pivotal role in this task, utilizing its capabilities to pinpoint the presence of faces within the frame. In cases where faces are detected near the periphery of the frame, a notification is triggered, prompting individuals to adjust their position and move closer to the frame's center. This adjustment is essential to ensure the effectiveness of the subsequent recognition process. On the other hand, in cases where more than one face is detected, the script

will prioritize the one nearest the center of the frame.



(a) YOLOv8 face detection



(b) FaceNet identification

Figure 39: Face Detection and Recognition process

The coordinates of the bounding boxes encompassing the detected face are then transmitted to the subsequent phase, where FaceNet assumes control. In essence, YOLOv8 acts as the locator, indicating the precise location of a face, while FaceNet takes on the responsibility of identifying the individual. The outcome of these intricately coordinated processes can be observed either on a monitor connected to the Jetson Nano or streamed through the Kafka ecosystem, as discussed previously. This integration ensures a comprehensive and efficient facial recognition system, capable of delivering real-time results and monitored from anywhere.

The development of an actual trigger function designed to log the attendance of present students and subsequently transmit this information to an IoT database through Fluentd is currently underway. However, owing to time constraints, the completion of this crucial component was regrettably delayed and could not be finalized within the designated timeframe.

10 Extension: License Plate Recognition

10.1 Extension Context

The versatile framework of the AIoT system offers a remarkable array of use cases, making it a robust choice for various applications. One particularly noteworthy application, as highlighted earlier, pertains to traffic monitoring, a domain with significant implications for safety, security, and efficiency.

The integration of License Plate Recognition (LPR) into this AIoT ecosystem represents a logical and powerful evolution of the system. By incorporating LPR capabilities, the AIoT system can extend its functionality to include the automated recognition and analysis of license plates on vehicles. This means that the system can efficiently capture, process, and interpret license plate information from images or video streams. Such capabilities open doors to an array of innovative applications, including tracking and identifying vehicles, managing parking lots, enhancing traffic flow, and bolstering security through improved access control.

10.2 EasyOCR

Figure 40: License Plate Recognition using EasyOCR

EasyOCR is an open-source Python library designed to perform Optical Character Recognition (OCR) tasks with ease. It is a powerful tool for extracting text and information from images, making it an invaluable resource in various applications, including license plate recognition. In this thesis, we will provide an overview of EasyOCR and explore its practical applications in the context of license plate reading.

EasyOCR is built on deep learning techniques and is capable of recognizing text in multiple languages with high accuracy. It stands out for its simplicity of use, offering a user-friendly

interface for developers to integrate OCR capabilities into their projects rapidly. This library provides a pre-trained model that can detect and extract text from images, including complex scenes with various fonts, sizes, and orientations.

Figure 41: Example of errors when using EasyOCR

In the initial phases of integrating EasyOCR into the system, the performance results, particularly when dealing with security webcam-quality images, have exhibited a mixed range of outcomes. At its best, the system manages to provide somewhat satisfactory results, deciphering text with an acceptable level of accuracy. However, at its worst, the accuracy of EasyOCR tends to drop significantly, rendering the captured text unreadable or entirely inaccurate.

This variance in performance can be attributed to several factors, including the quality of the input images, lighting conditions, and the inherent challenges posed by surveillance cameras. Security webcams often produce images that may be characterized by low resolution, suboptimal lighting, motion blur, and various other artifacts. These conditions present a formidable obstacle for any OCR system, including EasyOCR, as it attempts to extract meaningful information from such images.

So in the implementation section, a workaround is proposed.

10.3 Implementation

10.3.1 Isolated implementation

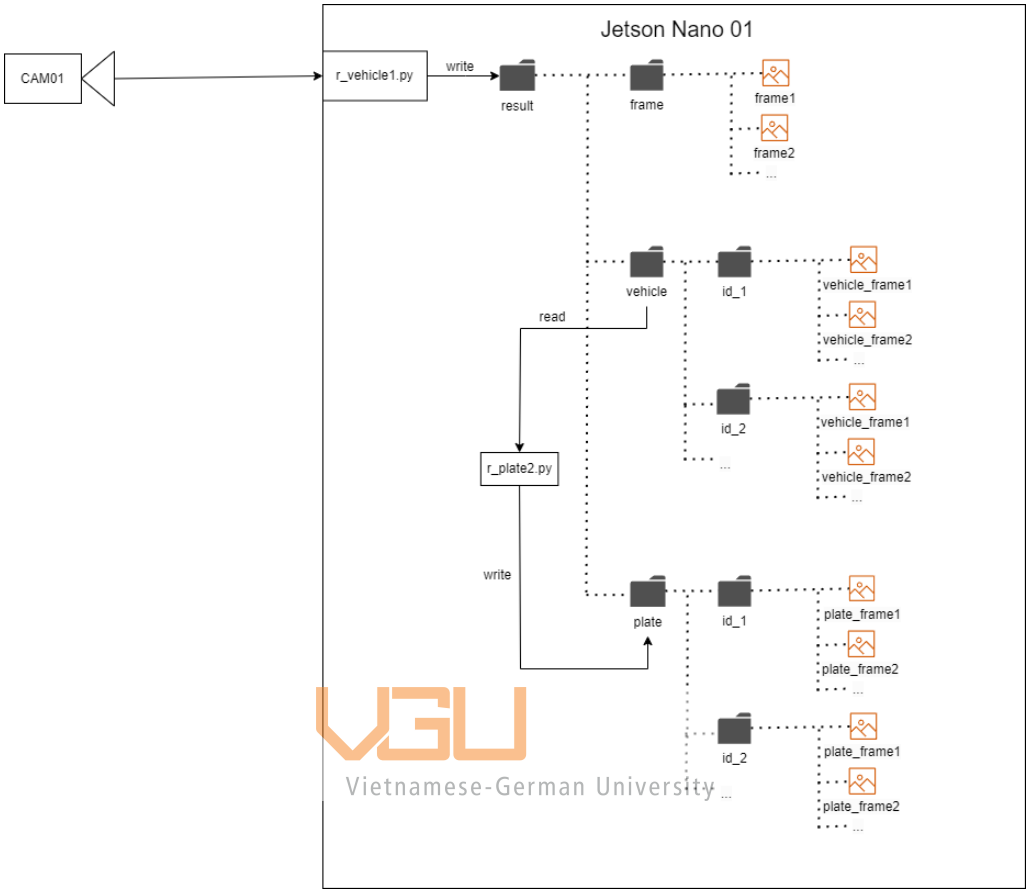


Figure 42: License Reader Implementation

This extension module plays a crucial role in the system by facilitating the extraction of vehicles from each frame within the input camera feed. Leveraging the advanced tracking capabilities of YOLOv8, this component efficiently identifies and tracks vehicles as they traverse through the video frames. The tracking process yields valuable coordinates, which are subsequently employed to isolate and extract individual vehicles from the footage.

Upon detection and tracking, each identified vehicle is precisely cropped and then systematically organized within designated folders. These folders are structured based on the unique *tracking_id* associated with each vehicle. The higher-level organizational framework entails the creation of a central directory labeled 'vehicle.' This directory, situated within the overarching 'result' folder, serves as the repository for the categorized vehicle subfolders.

In parallel with the vehicle data, the frames constituting the camera feed are also retained. These frames are meticulously preserved within the 'frame' directory, which is nested within the 'result' folder. The collective folder arrangement, as depicted in Figure 42, provides a coherent and accessible structure for storing and managing the data generated during the license plate recognition process.

Simultaneously, the system operates the 'r_plate2.py' script in parallel. This script is designed to autonomously navigate the 'vehicle' folder, a repository housing the cropped images of vehicles generated by the tracking module. Its primary objective is to perform license plate recognition for each identified vehicle using a custom-trained YOLOv8 module, a specialized component developed for this purpose.

The execution of this script serves a pivotal role in the system's workflow. It not only enables the precise identification of license plates but also establishes the crucial link between each license plate and its corresponding vehicle. Upon successfully detecting license plates within the cropped vehicle images, the system diligently organizes them into designated ID folders. These ID folders are thoughtfully structured and stored within the 'plate' directory, which is dynamically created within the overarching 'result' folder.

The ultimate phase of the process involves the utilization of EasyOCR or alternative OCR libraries to decipher the content of each license plate frame. To streamline these operations, these steps can be consolidated into a unified script and seamlessly reintegrated into the original system architecture. The primary objective is to extract and log the license plate numbers, subsequently forwarding this crucial data to the designated IoT database for comprehensive record-keeping and analysis.

10.3.2 Propose implementation and Integration

Nevertheless, it's worth addressing a noteworthy consideration arising from the performance evaluation of EasyOCR when applied to security camera feeds. In certain scenarios, EasyOCR's performance has proven to be suboptimal, ranging from modest accuracy to outright unreadability. To circumvent this limitation and ensure the reliability of license plate data, a pragmatic workaround has been devised.

The workaround involves a strategic shift in data processing. Instead of relying on text extraction from license plate frames, the system encodes and transmits the actual license plate frames themselves to the IoT database. This approach avoids potential inaccuracies or misinterpretations associated with text recognition. It also preserves the original visual data, ensuring that license plate information remains intact.

By opting for this methodology, the system effectively future-proofs its data collection process. This means that if the need arises for a more detailed or human-assisted analysis of license plate information, the original visual frames can be readily accessed and reanalyzed. This approach strikes a balance between real-time data processing and the preservation of high-quality, unaltered source data, bolstering the system's overall performance and versatility within the AIoT framework.

Another point of contention when it comes to LPR in Vietnam is the majority of vehicles are motorbikes. The license plate of a motorbike is situated in the back of the vehicle, unlike cars, which have one at each end of the vehicle. This makes reading the license plate of motorcycles moving toward the camera impossible. This means a modification to the established system is

required, with each camera monitoring the traffic lane moving away from the camera. Or better yet, the system is more suited for dashcam applications for LPR of traffic in front of an activated vehicle, instead of the static security camera approach.

An additional challenge in implementing LPR in Vietnam arises from the predominant use of motorbikes. Unlike cars, which typically feature license plates at both the front and rear of the vehicle, motorbikes have a single license plate located at the rear. This distinctive configuration poses a unique obstacle: when motorbikes are moving toward the camera, reading their license plates becomes an insurmountable task due to the plates' orientation.

To address this specific challenge and ensure the effectiveness of the LPR system in the Vietnamese context, it's imperative to consider necessary modifications to the established system. One plausible adjustment involves configuring each camera to monitor the traffic lane moving away from the camera. By focusing on the rear of the vehicles, where the license plates are positioned, the system can reliably capture and decipher license plate information.

Alternatively, a more tailored approach can be adopted by repurposing the system for dashcam applications. In this context, the system is ideally suited for real-time LPR of traffic situated in front of an activated vehicle. By installing cameras within vehicles and directing their field of view towards the road ahead, the system can adeptly recognize and log license plates from approaching vehicles. This dynamic approach aligns with the inherent mobility of dashcams, providing a versatile solution for on-the-go LPR tasks.

However, similar to Facial Recognition for Checking Attendance Extension, due to time constraints, these features were not researched further and implemented.

11 Conclusion

In conclusion, this thesis has explored the evolution and adaptation of an AIoT system designed for real-time monitoring and recognition tasks using the Jetson Nano. The system's journey reflects the dynamic nature of technology and its response to practical challenges and evolving requirements. Through various iterations, the system has undergone significant enhancements, each driven by a specific need or constraint.

The initial stages of the system's development centered on leveraging the capabilities of the Jetson Nano platform, where YOLOv8, a state-of-the-art object detection model, played a pivotal role. This phase laid the foundation for real-time object detection and monitoring, providing the basis for subsequent advancements.

As the project progresses, two noteworthy extensions are currently under active development, with one of them undergoing testing at VGU. These extensions represent promising additions to the AIoT system's capabilities. While the system remains in its prototype stage and functions as a working concept, its primary purpose is to demonstrate the utility and resilience of an AIoT system.

Even in this early stage, the system's architecture reveals substantial potential. It serves as a testament to the versatility and adaptability of AIoT technology. Beyond its immediate applications, the architecture hints at a myriad of possibilities for future enhancements and refinements.

Looking ahead, once the thesis period concludes, the project is poised for further development and refinement. The roadmap includes plans to transition from a prototype to a fully operational system. An integral part of this journey involves integrating the system into an active factory environment located in Dong Nai. This transition to real-world deployment presents a valuable opportunity to validate the system's effectiveness and robustness in an industrial setting.

Ultimately, the ongoing development and expansion of the AIoT system underscore its potential to offer tangible benefits across various industries. The commitment to improvement and real-world integration underscores the project's commitment to realizing the full spectrum of possibilities that AIoT technology has to offer.

References

[1] Loc, C., Do, N. (2023, June 3). AI Camera system for monitoring the Thai Ha crossing bridge, from https://video.vnexpress.net/tin-tuc/thoi-su/he-thong-camera-ai-bao-ve-cau-vuot-thai-ha-4612816.html?_gl=1

[2] MyAloha (2023, August 14). SECURITY GATE SOLUTION USING CAMERA FACEID, from https://myaloha.com.vn
and https://www.youtube.com/watch?v=60v3zolLi5s&ab_channel=MyAloha

[3] Vinpearl (2021, May 6). CHECK-IN BY FACIAL RECOGNITION TECHNOLOGY AT VINPEARL, from https://vinpearl.com/en/check-in-by-facial-recognition-technology-at-vinpearl

[4] NVIDIA (n.d.). NVIDIA Embedded Systems for Next-Gen Autonomous Machines. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/

[5] Enderle, R. (2022, January 18). Why NVIDIA Has Become a Leader in the AI Market. Datamation, from https://www.datamation.com/artificial-intelligence/why-nvidia-leader-ai-market/

[6] Sharon, G. (2023, February 23). How Nvidia dominated AI — and plans to keep it that way as generative AI explodes. VentureBeat, from https://venturebeat.com/ai/how-nvidia-dominated-ai-and-plans-to-keep-it-that-way-as-generative-ai-explodes/

[7] NVIDIA (n.d.). Jetson TK1 - eLinux.org. Elinux.org. Retrieved September 13, 2023, from https://elinux.org/Jetson_TK1

[8] Dmitry, S. (2023, February 12). A Guide to the YOLO Family of Computer Vision Models. Data Phoenix, from https://dataphoenix.info/a-guide-to-the-yolo-family-of-computer-vision-models/

[9] Ultralytics. (n.d.). Ultralytics. https://ultralytics.com

[10] thingsboard. (n.d.). What is ThingsBoard? ThingsBoard. Retrieved September 13, 2023, from https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/

[11] Things.vn. (2020, October 31). Full IoT solution for Viet Nam - Things.vn, from https://things.vn

[12] Aharon, N., Orfaig, R., & Bobrovsky, B.-Z. (2022). BoT-SORT: Robust Associations Multi-Pedestrian Tracking. ArXiv:2206.14651 [Cs]. https://arxiv.org/abs/2206.14651

[13] Zhang, Y., Sun, P., Jiang, Y., Yu, D., Weng, F., Yuan, Z., Luo, P., Liu, W., & Wang, X. (2022). ByteTrack: Multi-Object Tracking by Associating Every Detection Box. ArXiv:2110.06864 [Cs]. https://arxiv.org/abs/2110.06864

[14] Papers with Code - MOT17 Benchmark (Multi-Object Tracking). (n.d.). Paperswithcode.com. Retrieved September 13, 2023, from https://paperswithcode.com/sota/multi-object-tracking-on-mot17?p=bytetrack-multi-object-tracking-by-1

[15] Supervision. (n.d.). Supervision.roboflow.com. Retrieved September 13, 2023, from https://roboflow.github.io/supervision/

[16] Project, F. (n.d.). Fluentd | Open Source Data Collector. Www.fluentd.org. Retrieved September 13, 2023, from https://www.fluentd.org

[17] Bass, J. (2022, April 8). imageZMQ: Transporting OpenCV images. GitHub, from https://github.com/jeffbass/imagezmq

[18] Streamlit. (n.d.). Streamlit - The fastest way to build and share data apps. Streamlit.io, from https://streamlit.io

[19] MQTT. (n.d.). MQTT - The Standard for IoT Messaging Mqtt.org, from https://mqtt.org

[20] Eclipse Mosquitto. (2018, January 8). Eclipse Mosquitto. https://mosquitto.org

[21] Apache Kafka. (n.d.). Apache Kafka. https://kafka.apache.org

[22] YouTube (n.d.). YouTube Live Streaming API Overview. Google for Developers. Retrieved September 13, 2023, from https://developers.google.com/youtube/v3/live/getting-started

[23] NGINX. (2018). NGINX | High-Performance Load Balancer, Web Server, & Reverse Proxy. NGINX, from https://www.nginx.com

[24] Luka, D. (2019, November 15). Face Recognition with FaceNet and MTCNN. Arsfutura.com, from https://arsfutura.com/magazine/face-recognition-with-facenet-and-mtcnn/

[25] PyImageSearch. (2020, September 14). Getting started with EasyOCR for Optical Character Recognition. PyImageSearch, from https://pyimagesearch.com/2020/09/14/getting-started-with-easyocr-for-optical-character-recognition/