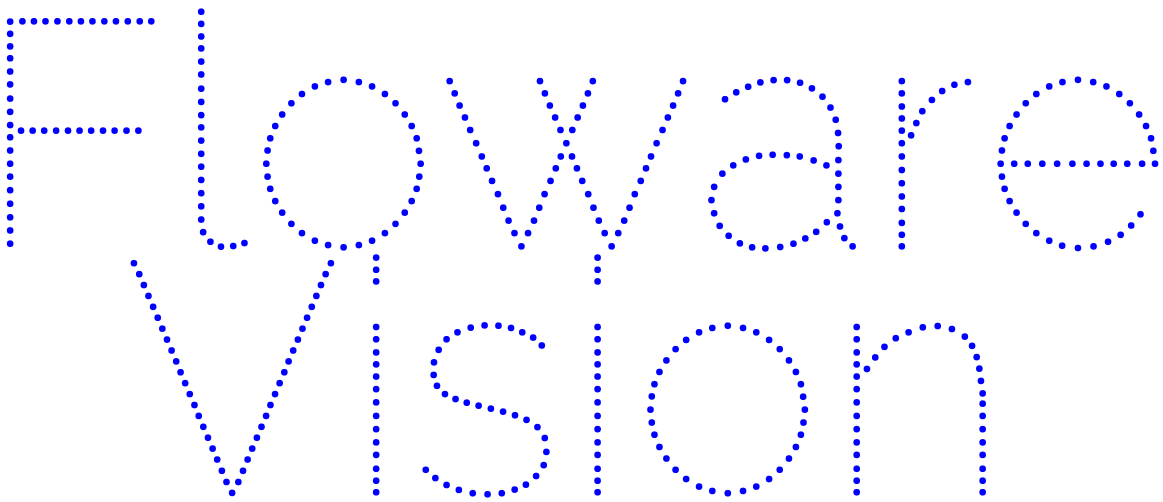


Journey in Edge AI Computer Vision Application for Traffic Analysis.

Noé Breton

Supervised by Julian Garbiso, Léonard Benedetti

EFREI Paris Panthéon-Assas Université

-

Floware

August 2023 ------> June 2024

Abstract

This memoir explores the development and application of Edge AI computer vision technology for traffic analysis through Floware Vision. The integration of computer vision, artificial intelligence, and the Internet of Things transformed various industries, transportation for instance. This project highlights Edge AI's real-time data processing capabilities to address mobility challenges and optimize traffic flow. Floware Vision, using Nvidia's Edge AI solutions and YOLO models, processes video inputs from multiple cameras to detect and track objects, providing essential data for traffic management. This approach not only enhances real-time decision-making but also maintains data privacy by performing computations locally. The memoir describes the architecture of Floware Vision, explaining the video processing pipeline, model integration, and the deployment strategies used. It aims to enhance the importance of modularity, real-time efficiency, and compatibility in the system's design. Additionally, it justifies the implementation of functional programming and microservices architecture to meet these requirements. Performance metrics and case studies comparing different YOLO models are presented to demonstrate the system's effectiveness. The memoir concludes with a review of the future work needed to further upgrade Floware Vision and its potential impact on urban mobility.

CONFIDENTIALITY

Plagiarism

I, Noé BRETON, hereby certify that I am the author of this Memoire, and I conducted the research myself. I confirm that this Memoire has not been previously submitted for any other degree. Any statement taken from the work of another person (with or without minor changes) and cited in this report is enclosed in quotation marks, and proper and precise references have been provided for such citations. I am aware that plagiarism can result in the invalidation of this Memoire and, in severe cases, lead to expulsion from the University. I also affirm that, except for the duly acknowledged citations, this report represents my own work.

Acknowledgements

I would like to thank Floware, Julian Garbiso, and Mathieu Lafarge for allowing me to build my experience in their company on interesting projects. Additionally, I would like to extend my thanks to all my collaborators, especially Ali, Bond, and Enea, for their support and sympathy.

Contents

Floware Vision: Journey in Edge AI Computer Vision Application Development for Traffic Analysis

Noé Breton

16 July 2024

1 Introduction

The convergence of computer vision, artificial intelligence, and the Internet of Things has revolutionized various industries like transportation, security, and home automation. Among these advancements, Edge AI has stepped up as a powerful and efficient solution for providing real-time data processing and decision-making capabilities directly at the source of data recording. Traffic analysis has benefited from these advancements. Indeed, the issue of *mobility flows*[1] represents a major challenge for many entities, whether public or private. According to a study of the *Institute for Transport & Economics Technische Universit¨at Dresde*, Traffic congestion typically leads to an increase in fuel consumption of about 80% while travel time can increase by up to a factor of 4[2].

Understanding and optimizing movements within a territory is essential for improving transportation efficiency, reducing congestion, and contributing to the transition toward more sustainable modes of transportation. In this context, the use of Edge AI sensors is really powerful as it can bring real-time processing and alerting by local computing and complex analysis by embedded AI software without privacy-threatening data transfers to the cloud. That is the path Floware has chosen. Floware seeks to provide useful information and innovative solutions to these entities over the long term, using Nvidia Edge AI solution and Yolo.

What role does Edge AI software play in traffic analysis ?

After providing some context by presenting the company and defining key notions in the field, with a significant state-of-the-art section on our utilized technology, my memoir will illustrate my contributions to Floware's activities, particularly on the embedded Edge AI software: FLWR-Vision. This approach will provide a comprehensive overview of Floware's activities, from the sensor to the data.

2 Floware: Edge AI for Traffic Analysis

2.1 Overview

Floware is a startup founded in 2023 in Palaiseau, specializing in analyzing mobility flows. It aims to bring innovative technologies for optimizing and de-carbonizing mobility over the long term. Relying on a network of proprietary sensors and a software chain strengthened by artificial intelligence, Floware aims to provide precise, secure, and anonymized data by controlling the entire data acquisition chain, from software to hardware, usable for a wide range of applications cases.

2.2 Team

Founded by Julian Garbiso (Telecom Paris Tech, IMT Atlantique), PhD in Computer Science and Networks, and Mathieu Lafarge (Science Po, ESTP), specialized in Urban Planning and Civil Engineering, Floware currently employs one trainee (myself) with the title of Edge AI and Cloud Services Developer. To strengthen our workforce and refocus my activity on sensor work, a recruitment campaign has been launched for two positions in Data Analysis and the Internet of Things. We now have a Senior Edge IA Engineer, *Enea de Bolivier*, (INP Grenoble, Kyoto University), an Industrial + UX/UI Designer, Daniel Esperben (Universidad Nacional de La Plata), and 3 interns. Ali Choucair (Université de Strasbourg), Mecatronics Engineer Intern, Bond Zhang (Polytechnique), Iot intern and Mathias (Epita), data scientist intern.

[1] Mobility flows refer to the movements and displacements of people or vehicles within a given space.

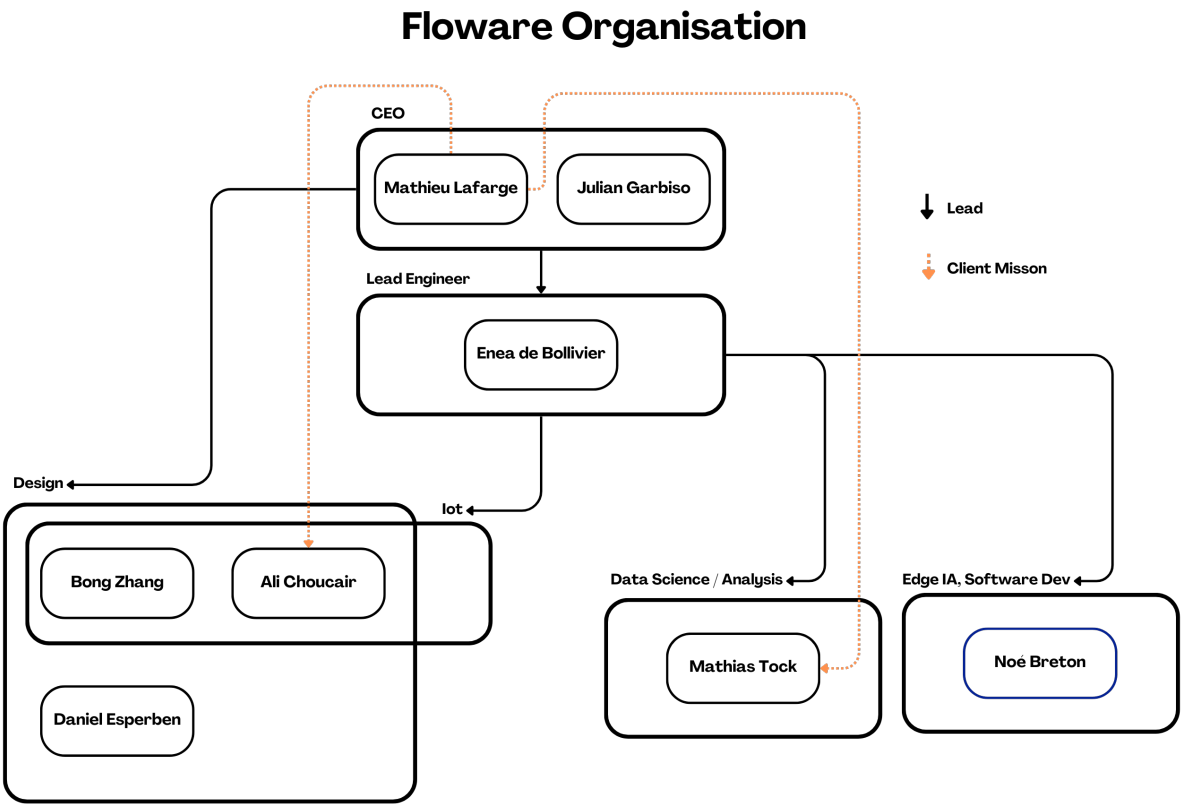


Figure 1: Floware Organisational Chart

2.3 Missions Organisation

- **Enea**
  - Manage Interns and trainees in all fields before new Seniors are hired.
  - Lead the embedded software part of Floware, including FLWR-Vision.
- **Ali & Bond**
  - Works on operational device deployment and hardware setup.
  - Research and development tasks on hardware (Motorized Camera, GPS feature, log writings)
- **David**
  - Design new device case improving temperature, weight, and size.
  - Design Floware websites.
- **Mathias**
  - Work on data science client missions: transform, and create a visualization of the data created from the sensor.

2.4 Partners [2]

This section outlines Floware's key partnerships, including academic, economic, and commercial collaborators. These partnerships play a crucial role in enhancing Floware's capabilities and market reach.

2.4.1 Academical Partners

Floware collaborates with several academic institutions to develop their technological expertise.

[2] Informations transmitten by Julian Garbiso

2.4.1.1 Institut Vedecom [3]

Floware is an affiliate of Vedecom Institute and is therefore a natural partner for R&D in the mobile sector. This partnership brings significant benefits in technical support, networking, and branding. Floware aims to create a mutually beneficial dynamic for both organizations by jointly responding to project proposals and tenders and developing a common communication and marketing strategy.

2.4.1.2 École Polytechnique (X-UP / X-TECH)

Floware was founded as a company as part of the X-UP acceleration program at the Ecole Polytechnique. Through the school's involvement in the company, the relationship has grown into a long-term partnership. Floware's offices are within X-TECH, allowing the company to continue benefiting from this valuable ecosystem's technological developments. The Drahi X-Novation Center also houses a FabLab, providing a conducive environment for Floware to develop and test its sensors on-site.

2.4.1.3 Université Gustave Eiffel

We are working with Mahdi Zargayouna, Deputy Director of the GRETTIA Laboratory, on Traffic Modeling and Simulation. The Sci-Ty program is currently developing a Technology Maturation and Transfer project.

2.4.2 Economic Partners

Floware's economic partners enhance the scalability of the business and support our business strategy, sales, and marketing.

2.4.2.1 HEC

Floware joined the HEC incubator in September 2023 and getting support in business strategy, sales, and marketing.

2.4.2.2 Moove Lab [4]

Floware is part of batch number 11 starting from September 2023. Its goal is to use this opportunity as a platform to access the ecosystem related to the mobility sector, including industry players and venture capital investors in the mobility market, notably ViaID, which runs the program with Mobilians.

2.4.2.3 Leonard (Groupe Vinci)[5]

Floware is part of batch number 11 starting from September 2023. Its goal is to use this opportunity as a platform to access the ecosystem related to the mobility sector, including industry players and venture capital investors in the mobility market, notably ViaID, which runs the program with Mobilians.

2.4.2.4 Nextmove [6]

Floware adheres to the Nextmove competitiveness cluster to benefit from its rich ecosystem, its network of member cities and territories, and its commitment to fostering a mutually beneficial partnership.

2.4.3 Commercial Partner

Floware's commercial partnerships focus on either implementing our sensor on their territory, providing useful data, or both.

2.4.3.1 Ecomesure [7]

Ecomesure is developing an IoT solution to measure air quality. Floware and Ecomesure are working together on EcoFlow, selected by Paris&Co as an innovative metropolitan area. The tool will monitor and warn about vehicle emissions to support low-emission zone policies in European cities.

[3] French Institute for Energy Transition dedicated to road mobility [3]

2.4.3.2 EPAPS (Etablissement d'aménagement de Paris Saclay)

Floware's first major partner commissioned them to conduct a large-scale proof of concept. EPAPS helped Floware to contract with EPAPS and DiRIF and collaborate with the Ecole Polytechnique to conduct decarbonization experiments.

2.4.3.3 La Fabrique des Mobilités [8]

Floware is working with FabMob to develop Tracemob, an app that tracks travel intentions. They plan to conduct a joint citizen travel survey in Noisy-le-Grand, combining sensor data and citizen voluntary travel information.

2.4.4 Comercial Partner Archetype

- **Local collectivities** (E.g, traffic data usage to optimize public transportation routes)
- **A developer/promoter** (E.g, real estate developers planning new residential areas based on traffic patterns)
- **A mobility operator** (E.g, bus companies adjusting schedules based on real-time traffic conditions)
- **A construction industry actor** (E.g, construction firms coordinating work schedules to minimize traffic disruptions)
- **An engineering firm** (E.g, firms designing new road networks with the help of advanced traffic modeling)

2.5 Location

- **École Polytechnique Palaiseau** - Drahi-X Novation Center
- **Station F** - Open Space Moove Lab, Paris 13th
- **Leonard:Paris** - Coworking space, Paris 11th

2.6 Some Competitors

- **Wintics**[9]: Develops AI-based software for traffic management, parking optimization and public transport monitoring to improve urban mobility and reduce congestion.
- **ALYCE**[10]: Provides insights into traffic patterns, pedestrian flows, and public transport usage, as well as mobility data and analytics to optimize infrastructure and services.
- **UPcity**[11]: Provides urban planning and smart city management tools, including traffic simulation and infrastructure planning, to support efficient and sustainable urban development.
- **Eurovia**[12]: VINCI Group subsidiary focuses on transport infrastructure construction and urban development, integrating IoT and AI to achieve sustainable and resilient infrastructure.
- **CDVIA**[13]: Provides advanced traffic management systems with real-time data analytics and machine learning to optimize traffic flows, reduce congestion, and improve traffic safety.

2.7 Services

As mentioned earlier, Floware offers an innovative solution for analyzing mobility flows for private and public territorial operators. Their philosophy is centered on privacy-by-design (privacy protection rooted in robust design) and focuses on efficiency and autonomy. Floware aims to control every aspect of the data pipeline, from the acquisition of data to its processing and analysis. Floware divides its services into three products :

Figure 2: Floware Solutions

2.7.1 Floware Vision

The embedded software that collects data and performs the first processing, powered by AI, is the subject of this memoir. You can find more details in the FLWR-Vision focused section ***here***.

2.7.2 Floware Core (In development)

SaaS platform for data analysis and visualization on the cloud. Production of mobility models and simulations tailored to specific use cases.

2.7.3 Floware Autopilot (In development)

Floware's AI co-pilot uses Larges language models to analyze data and manage cloud software. This tool provides clients with visualizations, reports, simulations, and customized insights. It improves strategic decision-making in mobility. Floware aims to be the top provider of these services with a scalable business model.

2.8 Project Management

Floware chose Agile management[14] for developing their solution, with long-term tasks and urgent, short-term tasks for clients' needs.

3 Edge AI: Definition and Context

3.1 Introduction to Edge Computing

After the deployment of the World Wide Web in 1989 by Tim Berners-Lee, and the invention of web servers, web browsers, and HTML, data processing started to switch from the local machine to a server. Berners-Lee noticed some issues: in the future, when many devices are connected to the internet and if all the data is processed by a group of centralized servers, congestion problems tend to appear, causing bugs and crashes for users. A decentralization of computation was needed. Akamai[15] was one of the first companies to introduce this concept in 1989. By using multiple networks physically closer to users and devices, latency can be reduced, costs minimized, and network connectivity improved. Imagine a festival with thousands of people inside. If there were only one big food stand, toilet, or bar, it would be a matter of time before the whole stand becomes bloated and nearly unreachable. Splitting the big stand into smaller ones, located in diverse places, and closer to the festival-goers that can get their food quickly and efficiently.

With time, edge computing extended to the content-delivering usage from other applications, and now defines every case where running a computer program delivers a quick response to where the request is made. Edge Computing is not the same concept

as the Internet Of Things (IoT), because this network of physical objects is not obligated to process the data, and can send their data to the cloud for processing, whereas edge computing focuses on local processing.

Edge Artificial intelligence (Edge AI)[16] is one of the derivate uses of edge computing. It refers to deploying models and running machine learning tasks directly close to the device location instead of a distant cloud. The data is stored and processed at the same place, before an optional sending to a cloud for retaining or deploying purposes.

3.2 Why using Edge AI ?

Cloud AI and Distributed AI are also popular solutions, but for some use cases, Edge AI brings distinct advantages, particularly for small structures operating in public spaces.

Operating at the device level removes the need to send data across the network, reducing the risk of data leaks or breaches[17]. Decentralization decreases the threat of major data breaches because data is not stored in a single location. Additionally, processing sensitive private data locally allows for retaining only the analysis output or encrypted information, minimizing privacy issues and improving **security**.

Additionally, on a **latency** aspect, by the edge computing properties mentioned above, the data is immediately processed without server traveling delay.

This reduced latency and local processing enables **instantaneous task treatment** and permits an alerting system. For instance, a Computer Vision IoT device can recognize a person searched by the authorities and immediately alert them in real-time.

Fewer requests to a distant server also mean **decreased bandwidth usage**, with all the cost reductions it induces. reduced bandwidth and cloud treatment enhance the **scalability** of edge solutions. Indeed, increasing the number of devices doesn't significantly increase the cloud resources needs because each device processes its data independently.

3.3 Applications of Edge AI

Edge AI can be applied to numerous domains. Indeed, edge AI devices like sensors, drones, and cameras can be used in **agriculture** to determine the health state of the soil, and infestation in real-time, allowing the farmer to act quickly, and reducing his work time[18].

Floware Sensors are a good example of edge AI utilization in **Traffic Flux Analysis**. Processing data directly from a camera stream lightens the data sent to the server while reducing the amount of sensitive information traveling to the server. Real-time processing enables an alerting process if congestion is detected, for instance.

In another case, a device with many diagnostic sensors can provide an immediate diagnosis with confidentiality. More softly, **health** monitoring with noninvasive edge devices like Fitbit[19].

4 Real Time Computer Vision and the YOLO approach

4.1 Introduction to computer vision

"Computer vision is a field of computer science that focuses on enabling computers to identify and understand objects and people in images and videos. Like other types of AI, computer vision seeks to perform and automate tasks by replicating human capabilities. In this case, computer vision seeks to replicate both the way humans see, and the way humans make sense of what they see"[20]

Computer vision regroups techniques and tools that enable computers to understand, interpret, and process information from images and videos. To understand an image, a computer translates different aspects of it into an array of numbers. These aspects can represent the contrast or color of each pixel, and the intensity of luminosity or the grey level, for example.

Computer vision existed before the rise of Deep learning and is still used in its traditional form for simple tasks, like edge or shape detection, and integrated into more complex processes. Convolutional Neural Networks, for instance, still use filters and shape detection in their process.

It was significantly transformed by two major developments: the invention of convolutions neural networks in 1980[21] and the rise of GPU Computing in the 2000s. But because of limited computer resources, it could not be used then. The latter was required as originally computing power was not sufficient to run CNN. In 2014, the CNN AlexNet trained on the dataset ImageNet dominated the image classification field and proved the efficiency of CNN on image classification. A CNN comprises layers that apply matrix operation at each step on the image vector. This method significantly improved the accuracy of object detection and enabled the handling of massive data volumes. Unlike traditional methods requiring manual feature engineering, Deep Learning algorithms can automatically learn relevant features from data, thus simplifying the development process significantly improving the accuracy of object detection, and enabling the handling of massive data volumes.

Despite the exponential growth of computer capacity, there were still performance issues, CNNs were requiring too many resources to run in real-time. But why it was still the case in the 2010s? Let's explain how real-time computer vision worked before 2016. A popular way to process vision object detection was through Faster RCNN (Fast Region-based Convolutional Neural Network). RCNN divides the image into smaller regions instead of grids, up to 2000 regions, and then passes these regions into a pre-trained AlexNet and a feature map, for each region before a regression.

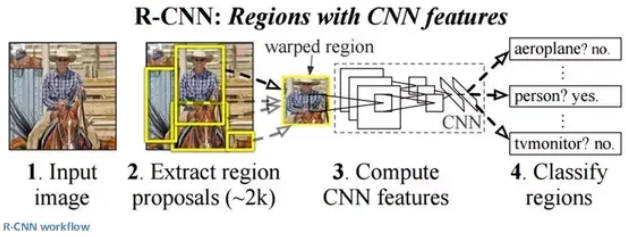


Figure 3: RCNN

Determining 2000 regions took too much time for real-time (47 seconds per frame), therefore fast RCNN was created, and this network established the region from the feature maps, without the need to feed the network 2000 regions directly.

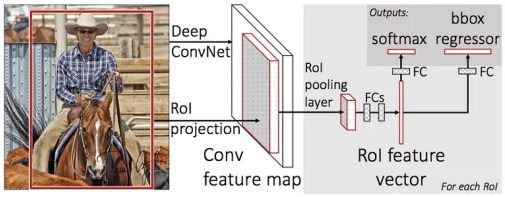


Figure 4: Fast RCNN

Faster RCNN improved the process by suppressing the selective search algorithm for regions and let a network predict them instead.

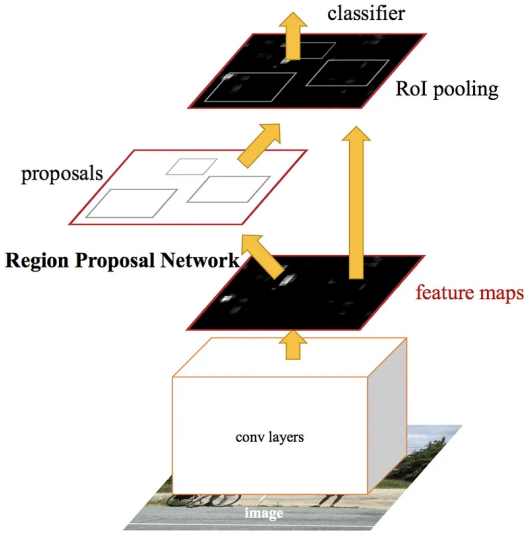


Figure 5: Faster RCNN

Despite the optimization improvements of these region-based models, it still wasn't enough for real-time use. To address these limitations, the YOLO[22] (You Only Look Once) approach was developed.

4.2 How YOLO Works

While RCNNs do not look at the complete image and process regions, YOLO chooses a different approach by processing one frame at once, using only one network. The network predicts bounding boxes and class probabilities directly from full images in one evaluation.

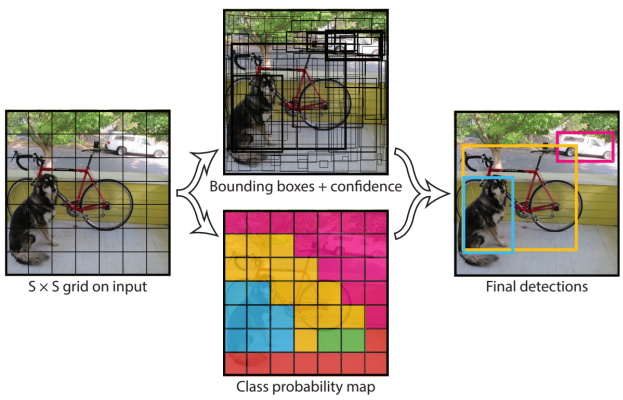


Figure 6: The Model

YOLO divides an image into an SxS grid of elements, and for each grid cell, it detects if an object is present through a multi-layer neural network. In the case of detection, it determines the center of the object (x, y), its height, width, and a class confidence score. The output is an object/tensor of dimension SxSx(B*5 + C).

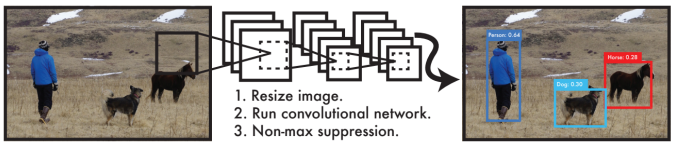


Figure 7: The YOLO Detection System.

4.3 YOLO's Iterations

YOLO has seen several versions, each improving after each iteration in terms of accuracy, speed, and capabilities. Here is a brief overview of the key iterations and advancements:

4.3.1 YOLOv1-v4

YOLOv1, which was first presented by Redmon et al. in 2016, transformed object detection by approaching it as a single regression problem. It enabled real-time detection by processing images on a GPU at 45 frames per second. In contrast to RCNN models, which were accurate but slow, YOLOv1 provided a noticeable speed boost.

YOLOv2, or YOLO9000, was established in 2017. Anchor boxes, batch normalization, and a brand-new network architecture known as Darknet-19 were all included. With over 9000 object classes detected, YOLO9000 improved speed and accuracy.

YOLOv3 in 2018, included detection at three different scales along with Darknet-53, an enriched architecture with residual connections. This version improved precision and recall while handling both small and large objects.

YOLOv4, released in 2020 combined new methods like PANet and SAM blocks with CSPDarknet53 as the backbone to optimize speed and accuracy. It enhanced the mean Average Precision (mAP) while retaining real-time detection.

4.3.2 YOLOv5-v8

YOLOv5, created by Ultralytics in 2020, was designed with deployment and usability in mind. Offering multiple versions suited to varying computational budgets, from mobile devices to server-grade GPUs, it was optimized for production environments.

YOLOv6, which was first introduced by Li et al. in 2022, was designed for industrial use. Its Rep-PAN neck, anchor-free detection method, and EfficientRep backbone optimized performance for particular industrial scenarios.

YOLOv7, published in 2022, combined novel model architectures with cutting-edge learning strategies. It created ”bag-of-freebies” methods to boost accuracy without raising inference costs, and it raised the bar for real-time object detection. YOLOv8, launched in 2023, kept improving performance and usability. It included mosaic data augmentation, anchor-free detection, and a decoupled head. With its effective real-time detection capabilities, YOLOv8 is especially well-suited for edge AI devices. Because of their smaller size and higher framerate, YOLOv8 in its small (YOLOv8s) and nano (YOLOv8n) versions are used on Floware embedded computer vision.

4.4 Comparison with Competitor Models

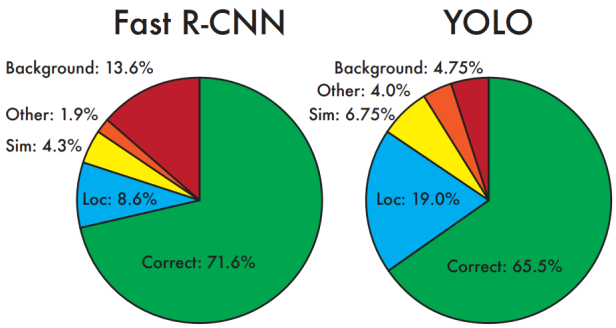


Figure 8: Error Analysis: Fast R-CNN vs. YOLO : Despite R-CNN having more correct detection, Yolo compensated by better performances

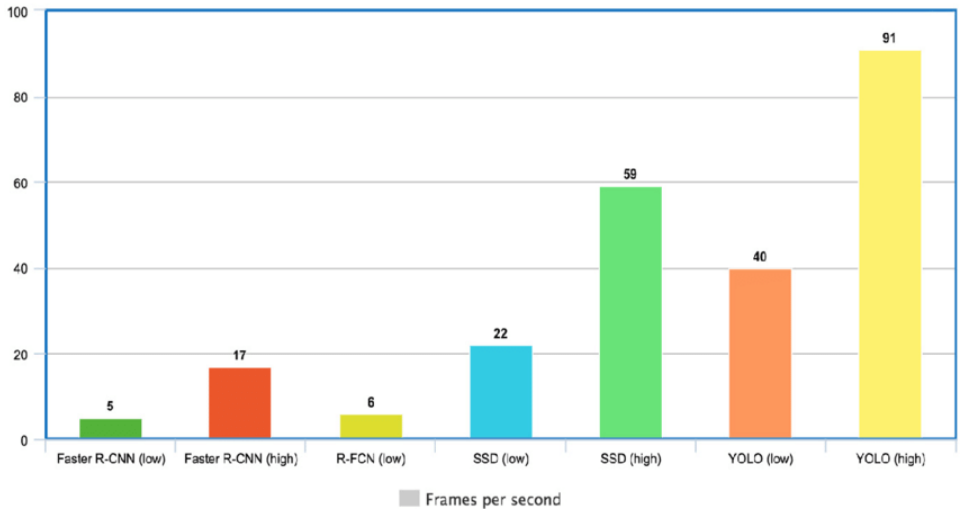


Figure 9: YOLO Framerate compared to other state-of-the-art object detectors [23]

4.5 YOLO limitations

While YOLO is a powerful object detection algorithm, it suffers from some limitations:

YOLO has several limitations that affect its performance in certain applications. One major limitation is the detection of small objects within large images due to its grid-based approach, which can lead to inaccurate predictions. Additionally, YOLO may struggle with objects that are too close to each other, causing overlapping bounding boxes and reduced detection accuracy. The algorithm also tends to underperform in scenarios with significant variations in object scale, orientation, and occlusion. YOLO can also be less precise than real-time detectors such as Faster R-CNN, even though this issue is compensated by its real-time processing capabilities. YOLO does not include tracking functionality natively. These limitations must be considered when applying YOLO to traffic detection tasks, where high accuracy and reliability are crucial.

4.6 Application of YOLO in traffic detection

Despite its limitations and coupled with tracking algorithms, YOLO has been effectively applied in traffic detection systems due to its real-time processing capabilities. Its speed allows for quick identification of vehicles, pedestrians, and other objects, making it suitable for dynamic traffic environments. YOLO can be integrated with edge AI hardware to analyze video streams from traffic cameras, processing data for traffic management and monitoring. By optimizing the model and fine-tuning it for specific traffic scenarios, YOLO can achieve satisfactory performance levels, contributing to efficient traffic flow and incident detection (see figure 12).

5 Nvidia Jetson: The Chosen Hardware

5.1 Introduction to Nvidia Jetson[1]

The Nvidia Jetson is a family of hardware designed for edge and embedded computing. These devices are engineered to efficiently run AI and machine learning models, using GPU computing and Nvidia technology, such as the DeepStream SDK, enabling developers to create AI systems at the edge. This makes them ideal for various applications, including autonomous machines, IoT devices, and edge computing solutions. Nvidia offers different versions of Jetson with varying prices and functionalities: Jetson Nano, Jetson Orin, and Xavier listed in ascending order of capabilities and prices. Currently, Floware uses the Nano and Orin models due to their scalability and cost-effectiveness. However, Nvidia has discontinued the Jetson Nano due to its aging performance capabilities, forcing Floware to transition to the Jetson Orin.

5.2 Technical Specifications

Characteristics	Jetson Nano Dev Kit	Jetson Orin Dev Kit 8GB	Raspberry Pi 4 + Coral TPU
Processor	+ Quad-core ARM Cortex-A57 @ 1.43 GHz (decent performance)	+ 6-core ARM Cortex-A78AE @ 2.0 GHz (very high performance)	- Quad-core ARM Cortex-A72 @ 1.5 GHz (faster)
GPU	+ 128-core Maxwell integrated GPU	++ 1792-core Ampere integrated GPU	- VideoCore VI (less performant without TPU)
AI Accelerator	+ Integrated	+ Integrated (JETSON Orin: 200 TOPS)	- Requires addition of Coral Edge TPU (4 TOPS)
Memory	- 4 GB LPDDR4	+ 8 GB LPDDR5	- Options of 2 GB, 4 GB, or 8 GB LPDDR4
AI Performance	- Up to 472 GFLOPs	+ Up to 200 TOPS	+ Coral TPU: 4 TOPS (superior AI performance)
Connectivity	+ 4x USB 3.0, HDMI, DisplayPort, Ethernet	+ 4x USB 3.2, 2x USB-C, HDMI, DisplayPort, Ethernet	- 2x USB 3.0, 2x USB 2.0, HDMI, Ethernet
Supported OS	+ Ubuntu, JetPack SDK (incl. TensorRT)	+ Ubuntu, JetPack SDK (incl. TensorRT)	- Raspberry Pi OS, Ubuntu, other Linux distros
Ease of Use	+ Simplified configuration with JetPack SDK	+ Simplified configuration with JetPack SDK	- Initial configuration more complex with Coral TPU
Cost	- More expensive than Raspberry Pi 4 alone (100€)	- More expensive than Jetson Nano and Raspberry Pi 4 (500€)	- Combined cost (Raspberry Pi + Coral TPU) can be higher than Jetson Nano (140€)
Advantages	+ Native GPU integration, robust software support	+ Extremely high AI performance, robust software support	+ Memory flexibility, high AI performance with Coral TPU
Disadvantages	- Limited performance compared to more advanced models	- Higher cost	- More complex configuration, performance management more difficult
Best Choice for YOLOv8	+ Advantages	++ Advantages	- Disadvantages

Figure 10: Specification comparison between Jetson Nano, Orin and a Raspberry Pi

5.3 Applications in Floware Vision

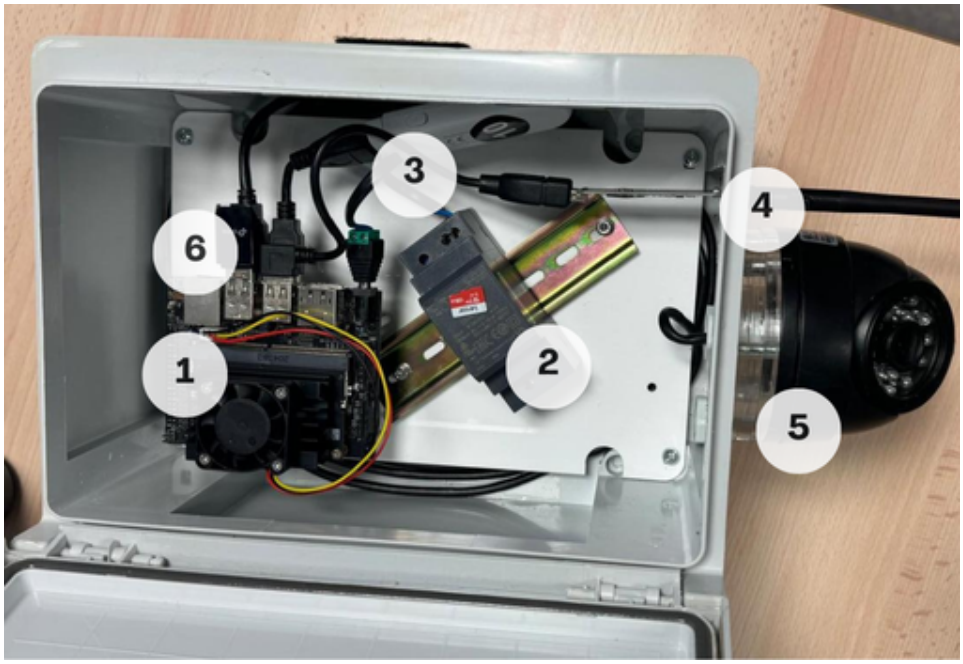


Figure 11: Floware Sensor

A classic Floware sensor is composed of a Jetson Nano (**1**) linked to a camera (**5**), and an Ubertooth antenna [24] (**4**) to scrape Bluetooth data. The Jetson Nano does not contain a Wi-Fi card, so we use a Wi-Fi dongle (**3**) with a Wi-Fi key (**3**) to connect to the network. The Nvidia Jetson Nano is powered by a 12-volt to 5-volt converter(**2**).

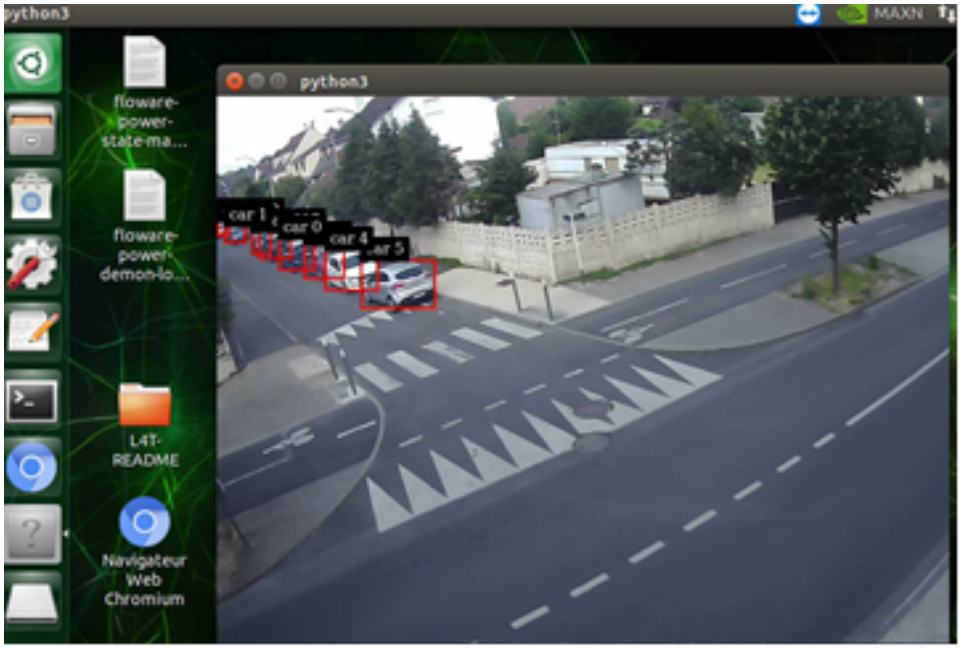


Figure 12: Jetson Nano Desktop with Floware Vision Running

6 Focus On Nvidia Deepstream, the complexity worth the performance ?

Deepstream is an Nvidia-developed project based on Gstreamer, an open-source pipeline-based framework [25] made for processing media streams. Gstreamer links many stream-processing into complex workflows.

6.1 Overview of Nvidia Deepstream

Nvidia DeepStream[26] reuses tasks in a cascade routine but is specialized for multi-sensor (audio, video, and image) AI processing. DeepStream incorporates deep neural networks and other complex processes such as encoding/decoding, rendering, and tracking. GStreamer's basic pipeline element is still usable, in addition to Nvidia's created element plugin. DeepStream can be used on multiple devices, personal computers with Ubuntu installed, or with Nvidia edge devices, such as Nvidia Jetson, used for Floware Sensor.

6.2 Features and Capabilities

- **Multi-Stream Processing**: Capable of managing multiple video streams at once, ensuring efficient use of resources.
- **AI Models Integration**: Allows modular integration with multiple pre-trained AI models for detection, classification, and segmentation.
- **Hardware Acceleration**: Permits full integration with Nvidia graphics card acceleration. GPU enables fast parallel computation, improving performance. Every element of the pipeline is GPU computed.
- **Scalability**: Adaptable for deployment on various platforms, from edge devices to cloud environments, supporting scalable solutions.
- **Customizability**: Offers a flexible pipeline architecture that enables easy fine-tuning of complex processes to fit various application needs.
- **Developer Tools**: Includes tools for performance monitoring, debugging, and optimization to assist developers in fine-tuning their applications.
- **Compatibility**: Deepstream is adaptable on many platforms, from edge devices to cloud environments.

6.3 Performance vs. Complexity

Setting up and optimizing DeepStream requires a deep knowledge of the Nvidia ecosystem, including CUDA and TensorRT. Customizing and fitting a pipeline for our needs can require more development and testing effort. Why would we use DeepStream for a computer vision task instead of a basic Python script using Ultralytics[27] and OpenCV on GPU? Besides the fact that the Jetson Nano was built to use DeepStream, the customizability and scalability arguments, an OpenCV application is never entirely running on the GPU, such as the rendering of videos and bounding boxes. In contrast, a DeepStream pipeline runs entirely on the GPU, allowing for overall better performance.

7 Floware Vision: Review of the Existing

Floware Vision, as discussed in the first part of the report, is the embedded software used in the Floware sensor. Floware Vision aims to centralize all the functionalities and AI processes needed for the output data, using Deepstream and Ubertooth as their main dependencies to perform computation and processing on the edge. Floware's needs include object detection, tracking, and classification (not in real-time but very regularly).

Additionally, Ubertooth is used to capture a fragment of the Bluetooth address (LAP) of the Bluetooth devices emitting packets in the detection range of the sensor, anonymizing it on the fly, helping in the continuity of detection between sensors. For some clients, relay control is required. As an example, local authorities may need to trigger lights when a person passes by during the night but not for a vehicle. This requires a relay control in the object detection pipeline. If the sensor camera angle is wide, triggering should be limited to a particular zone, such as a pedestrian crossing. This application needs a Region Of Interest (ROI) filtering to limit detection to a small zone, necessitating an ROI drawing tool and an encoding convention (either on the edge or in a preprocessing step).

This example highlights that the sensor has to be adapted to different use cases. Modularity is crucial to fine-tune each sensor according to the client's needs while responding to privacy needs.

As many sensors can be deployed simultaneously, they must be as failure-proof as possible. Floware needs to incorporate features that ensure reliability, and if a failure occurs, it is essential to understand why and how it happened.

7.1 Existing States

Floware Vision is a composite application, using a corpus of Python scripts distributed across multiple files. It manages Bluetooth data, computer vision data, but also some intern information like power and disk usage data.

7.1.1 Inputs

- **Video Stream via USB camera.**
- **Bluetooth data via Ubertooth antenna.**
- Disk usage.
- Date and time.

7.1.2 Outputs

Floware Vison sends It's output to servers (virtual machines) on the Scaleway or Azure clouds.

7.1.2.1 Vision Detection Data

Sensor output data is in the form of a CSV or Parquet file, divided by variable periods, with the following columns: `['Frame_number', 'ObjectID', 'ClassID', 'Confidencelevel', 'BoundingBoxTop', 'BoundingBoxLeft', 'BoundingBoxWidth', 'BoundingBoxHeight', 'Datetime']`

- `ObjectID:` corresponds to a unique identifier for a unique object.
- `ClassID:` corresponds to the type of object among these categories: `('car', 'truck', 'bike', 'motorbike', 'person')`
- `BoundingBox_:` corresponds to the characteristics of the detected object's frame.
- `Confidencelevel:` corresponds to the YOLO class confidence.

Each file is dated.

7.1.2.2 Snapshot Data

Snapshot data correspond to frames of the camera stream captured at certain intervals in JPG format. There are two categories of snapshots: time and class snapshots. The snapshots taken by class are named according to the detected class in the picture. The purpose of snapshots is to aid in debugging in case of incoherent results, provide an accurate representation of the sensor situation (snapshot time), and build our internal dataset (snapshot class).

(a) Example of Snapshot Class: truck-20240423165514

(b) Example of Snapshot Time: snapshot-202405032204s

7.1.2.3 Bluetooth data

Bluetooth data, collected using the Ubertooth[24] device, is sent to either the log or parquet format for further processing. The collected data includes several key fields, which are as follows: `['systime', 'ch', 'LAP', 'err', 'clkn', 'clk_offset', 's', 'n', 'snr']`. Each field represents specific information:

- `systime`: The system time when the data was captured.
- `ch`: The channel on which the data was captured.
- `LAP`: The Lower unique Address Part of the Bluetooth devices address. It's the main information for FFloware's Bluetooth data analysis.
- `err`: Error information, if any, related to the data capture.
- `clkn`: The clock number of the Bluetooth device.
- `clk_offset`: The clock offset.
- `s`: The signal strength or quality.
- `n`: The noise level at the time of data capture.
- `snr`: The signal-to-noise ratio, is useful for knowing the clearness of the signal.

This structured format ensures that the data can be effectively analyzed and utilized for traffic analysis and other applications.

7.1.2.4 The ippadrr case

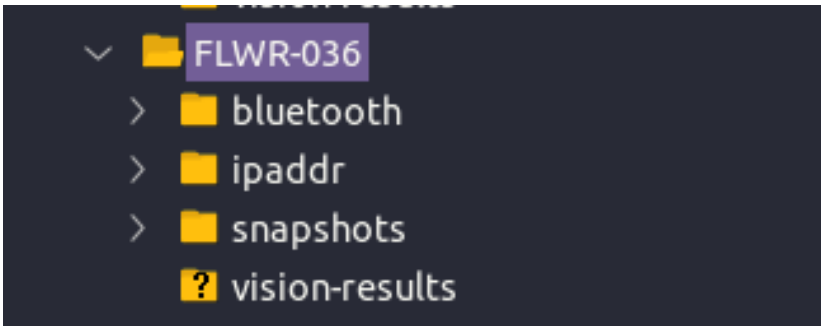


Figure 14: Server Side Folder Tree

The `ipaddr` folder is deprecated. It was initially created to regularly send the IP address information of the sensors (on the cellular network) to establish SSH connections. However, there were issues bypassing the NAT of the 4g dongle; thus, it was abandoned. Eventually, it was repurposed to transfer occasional files, such as syslog files. This functionality should be retained, but it should not be called `ipaddr`.

7.1.3 Key Features

Computer Vision (FLWR-Vision.py)

- Implementation of the Computer Vision module can be found in `FLWR-Vision.py`. This script configures and executes a GStreamer pipeline to process video streams. It offers primary and secondary inference, tracking, and analytics features using DeepStream SDK.
- Additionally, the module contains a scheduler that periodically converts CSV files to Parquet format, enabling efficient storage and analysis of the generated results.

Bluetooth Detection Scrapping (`FLWR-UbertoothService.py`)

- Responsible for capturing Bluetooth data using the Ubertooth device which executes the `ubertooth-rx` command and records the received data.
- Periodically (every 10 minutes), the script generates new log files to facilitate easier management and analysis of the data by segmenting it based on time intervals.

Server File Sender (`FLWR-SyncFoldersDaemon.py`)

- The `FLWR-SyncFoldersDaemon.py` script synchronizes local folders with remote server folders using the `rsync` command.

ROI Filtering (drawRoi.py)

- The `drawRoi.py` script facilitates the drawing of Regions of Interest (ROIs) for the analytics module.

Logs to Parquet Converter (`FLWR-UbertoothLogsToParquet.py`)

- The `FLWR-UbertoothLogsToParquet.py` script scrap log files generated by the Ubertooth device, converting them to Parquet format for optimized storage and analysis.
- The script parses each log file, extracts the relevant fields and saves the data into Parquet files. After successful conversion, the original log files are deleted to save disk space.

Disk Usage Control (`FLWR-DiskUsageControlDaemon.py`)

- The `FLWR-DiskUsageControlDaemon.py` script monitors disk usage in specified directories and deletes the oldest files when usage exceeds predefined limits.
- It ensures that the system remains operational by preventing disk space from being filled up, thus avoiding potential crashes or data loss.

Power Management (`FLWR-PowerDaemon.py`)

- The `FLWR-PowerDaemon.py` script manages the power state of the system, including suspending during night hours and rebooting in the morning.
- It uses a state machine approach to track the current power state and performs actions based on the time of day, ensuring optimal power usage.

7.1.4 Architectures

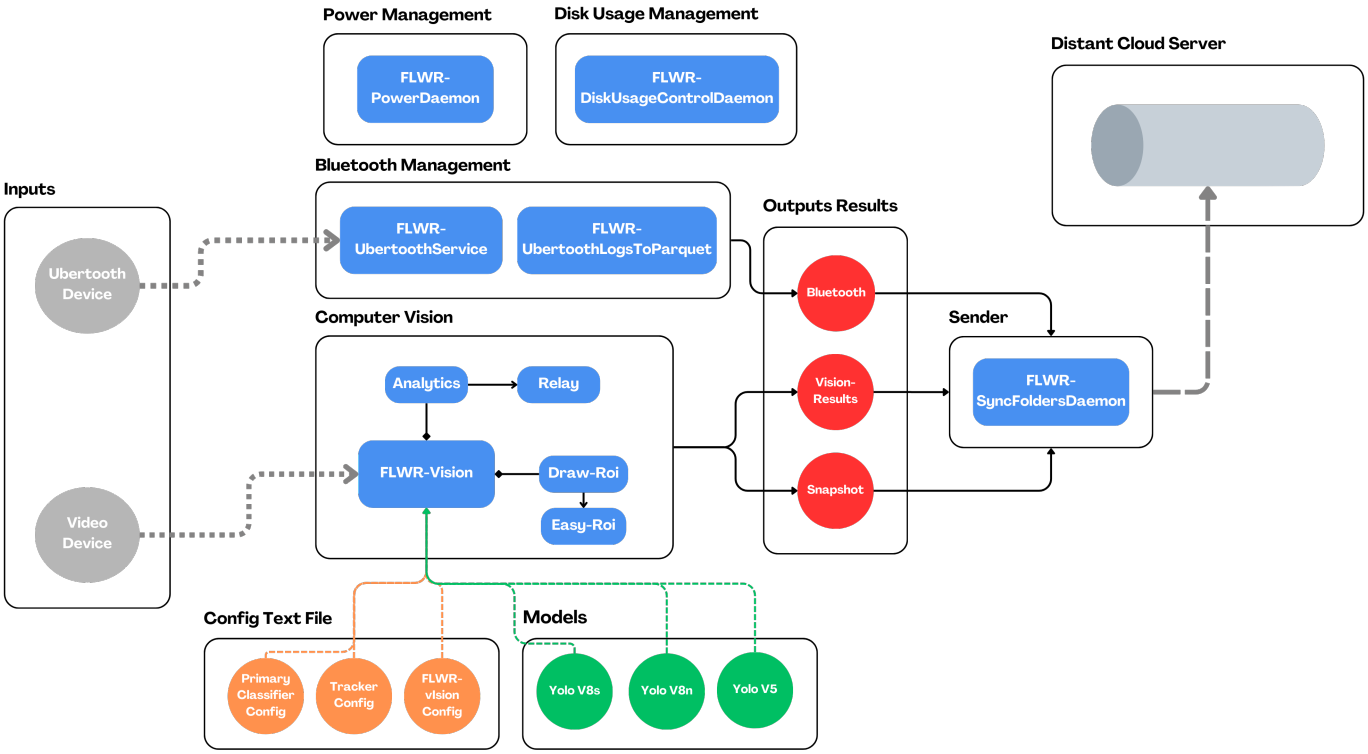


Figure 15: Floware Vision Architecture Representation

All the real-time processing is managed by the Analytics class, which saves computer vision results and activates the relay depending on the class detected.

7.1.5 Vision Processing Pipeline

As mentioned, FLWR-Vision uses a Deepstream pipeline to process video flux for object detection, tracking, and analytics. The pipeline is implemented in the `FLWR-Vision.py` script and uses the DeepStream SDK, enabling GPU processing throughout the entire pipeline.

Pipeline Components

- **Source Element**: The pipeline starts with a source element that captures video from available video devices. The function `list_video_devices()` is used to list all video devices available on the system.
- **Inference Engine**: One of the main elements of the pipeline is the primary inference element. It uses pre-trained models like Yolo to perform object detection and classification.
- **Tracker**: The tracking element tracks detected objects across frames. It attributes a unique id for each detected object.
- **Analytics Module**: An analytics module processes information output from the tracking and inference elements, performing analysis if needed.
- **Sink Element**: The sink element is placed at the end of the pipeline, which can either display the video on the screen or save it to a file, depending on the configuration.

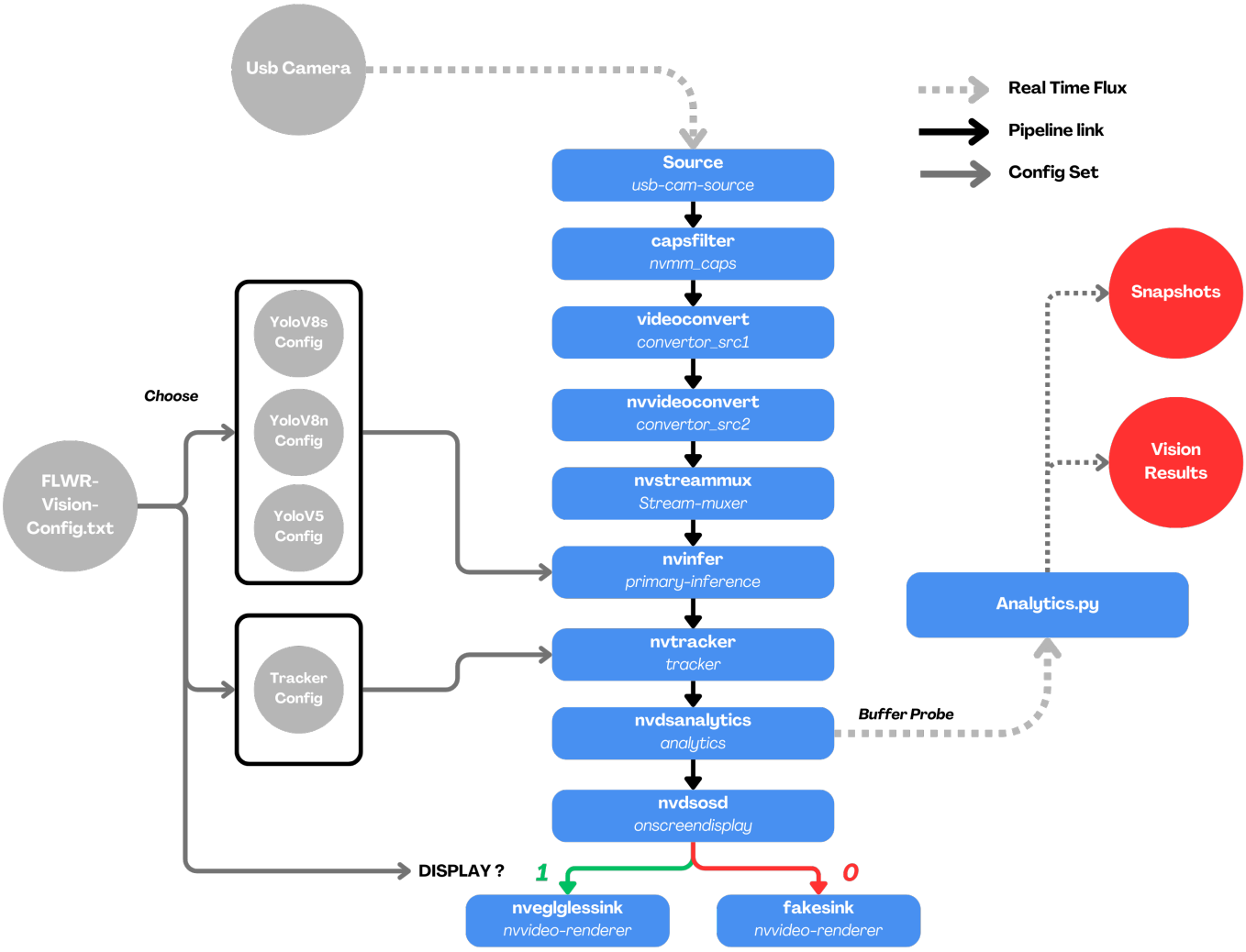


Figure 16: Existing Deepstream Pipeline Representation

Pipeline Elements

The `usb-cam-source` element captures video input from the USB camera, serving as the starting point of the video stream in the pipeline. Next, the `nvmm_caps` element applies properties to the video stream to match the expected format for future processing, setting up properties like frame rate, resolution, and format. Following this, the `convertor_src1` element converts the video stream into a compatible format for the next pipeline elements, ensuring compatibility for hardware-accelerated GPU processing. Another video conversion is performed by `convertor_ssing`. After that, the `Stream-muxer` element concatenates multiple input video streams into a single output stream, which is essential for scenarios where multiple camera feeds are processed simultaneously. Then, the `primary-inference` element performs primary inference using a pre-trained deep learning model, such as YOLO, and is responsible for object detection and classification within the video stream. Configurations for YOLO models (YOLOv8s, YOLOv8n, YOLOv5) can be loaded via the config file here. Additionally, the `tracker` element tracks detected objects across frames, maintaining unique identifiers for each object, and ensuring continuity of object detection over time for one sensor, which is essential for traffic detection. The `onscreendisplay` element then overlays information such as bounding boxes and labels on the video stream, useful for visualizing detection and tracking results in real-time, mostly for debugging and maintenance. Following this, the `nvvideorenderer` element renders the processed video stream to a display, providing real-time visualization of the pipeline output. A sink element, `fakesink`, does not display the stream visualization and is used when visualization is not needed; it is not currently used in production. Lastly, the `analytics` element performs additional analytics on the inferred data, processing the detection and tracking results to generate insights and metrics. In Floware Vision, the analysis part is transmitted to the Analytics class.

7.1.5.1 Overall Flow

- Video is captured from the USB camera.
- The video stream is processed through several conversion steps to ensure compatibility.
- Inference is performed to detect and classify objects.
- Detected objects are tracked across frames.
- The results are displayed on-screen or sent to analytics for personalized processing.
- Additional analytics are performed to generate insights and results are saved.

7.1.6 Model Integration

At the primary inference level, **YOLOv8s** is used by default and pretrained for 640x480 images. Labels used (see Vision Data Part is tuned for traffic purposes), for performance purposes YOLO8n can be used for improving framerate (see Floware Vision Performance Metrics).

7.2 Existing Deployment Process

Systemd services are used to maintain scripts to run even after failure.

7.2.0.1 Jetson Nano

On the Jetson Nano, Floware uses an SD card already flashed that contains Floware Vision from a previous Floware Sensor, then copies bit by bit data on another sd card to avoid most of the failure that can happen on a regular copy. The new sd card is then inserted in a new jetson. Bit-by-bit copy is a long process and can be time-consuming, it also does not allow any automatic Floware Vision Update.

7.2.0.2 Jetson Orin

Previous deployment on Jetson Orin is slightly different because it's a "recently" acquired hardware. Floware chose an independent developer to convert floware Vision to Orin. The independent developer created docker images that contain every dependency that Floware Vision requires. By pulling the docker images in a "virgin" Orin with Floware Vision on it, we can launch the application inside the docker. I was far from being finished at this state.

7.3 Limitations, Issues & Challenges

Checking the Floware Vision codebase revealed that a lot of angles of improvement were possible.

7.3.1 Structure Issue

Floware Vision's codebase doesn't have code documentation, making it very difficult to understand the purpose of each functionality. Besides the lack of documentation, there are many unused files with no purpose in production, including various sh, txt, and py files.

Additionally, the class used in the code has issues that make it hard to read. The main script, `FLWR-Vision.py`, is not structured with methods or classes, which is a major issue for modularity. If we wanted to add elements to the pipeline, such as a second classifier, or fix any bugs, every feature could be impacted by small modifications.

Even if a configuration file rigs this script, it still lacks modularity, and many features are hardcoded. For instance, we cannot choose the video device or the path of the configuration file for the tracker and the primary inference; only the model name is in the configuration. The script uses an if-else structure with hard coded paths inside the code:

```
pgie = Gst.ElementFactory.make("nvinfer", "primary-inference")
if not pgie:
    sys.stderr.write("␣Unable␣to␣create␣pgie␣\n")
if (c_app['app']['MODEL'] == 'YOLOV8N'):
    pgie.set_property('config-file-path', "cfg/YOLOV8N.txt")
elif (c_app['app']['MODEL'] == 'YOLOV8S'):
    pgie.set_property('config-file-path', "cfg/YOLOV8S.txt")
```

```
elif (c_app['app']['MODEL'] == 'YOLOV5N'):
    pgie.set_property('config-file-path', "cfg/YOLOV5N.txt")
elif (c_app['app']['MODEL'] == 'YOLOV5S'):
    pgie.set_property('config-file-path', "cfg/YOLOV5S.txt")
Else:
    print('Model Not found')
```

7.3.2 Security Issues

The previously reported structural issue also brings major security concerns. The script responsible for synchronization with the server has the server IP hardcoded. Additionally, there were two scripts with the same code but with different IPs for Scaleway and Azure, indicating a lack of modularity. The authentication keys are stored in the main Floware Vision repository without any protection. If an intruder accesses one of the Floware Jetson devices, they will also gain access to our servers. The Docker image of the application was publicly pulled from the personal DockerHub of a freelance developer.

7.3.3 Optimization

Floware Vision on Jetson Nano runs at 8 fps, while on Jetson Orin it runs at 30 fps. The low resolution and frame rate can reduce the precision of the measurements. There is also a size optimization issue; the Docker image for the Jetson Orin version of the application is 16 gigabytes, which is unusually large as it runs directly on the disk, indicating that the application is not properly containerized.

7.3.4 Deployement issue

The deployment method on the Jetson Nano does not permit efficient software updating on the deployed sensor. For example, we used the TeamViewer remote file transfer system to update a dozen sensors. An operation like this usually takes one and a half days, and many issues can occur during the process, such as ownership problems with the executable file or file conversion issues if the source computer is running Windows. Additionally, there was no online repository for the application, such as GitHub or GitLab, nor any version control.

7.3.5 Bugs

Some bugs were still in production and slowed the maintenance and the post-process offset:

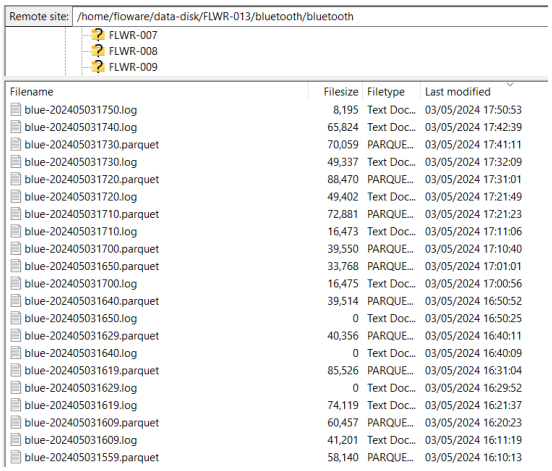
- Permission issue: The server key was not always accessible for the system's service due to an ownership issue.
- Files sending issue: On the server side, files were sent as soon they were written inside, so a lot of data was missing, they were also sent before the conversion to parquet, sometimes with only an error output inside.
- As the video device name was hardcoded in the source pipeline element (/dev/video0), the video device was not always at this location, and a manual reboot was required.

7.4 Feedback and Improvements

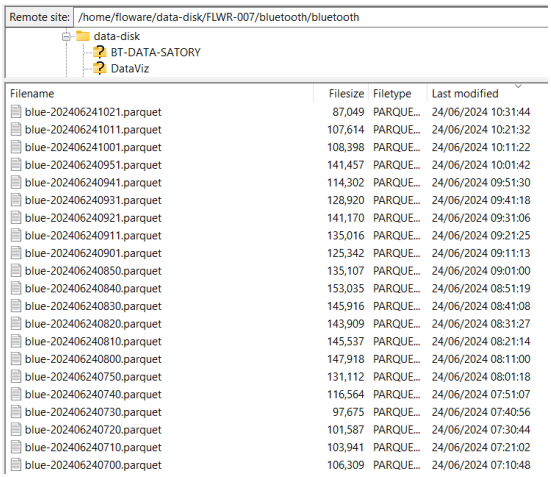
All these issues highlight the need for structuring and organization of a work pipeline so that each fix and addition can be reviewed and homogenized. Besides bug fixes and performance improvements, an effort will be made in modularity and compatibility to ensure efficient debugging and adaptability for various missions, particularly in AI detection. Indeed, some missions require fine-tuning in detection, such as work vehicle detection, license plate detection, or even speed detection. The future development and deployment process of Floware Vision will be considered in this way.

8 Floware Vision: Development

Before any refactoring, correcting the failure-causing bugs was the priority. First, we created a bash script that deployed the system's services as the IoT user, eliminating permission issues. In the file-sending system services, we chose to ignore the last file created and exclude any logs and CSV files. The video device selection was no longer hardcoded; we retrieved a list of all devices and selected the first one from the sorted list.



(a) Bluetooth Data before fixes



(b) Bluetooth Data after fixes

The Floware Vision development and refactorization process is driven by three constraints: **real-time efficiency**, **modularity**, and **compatibility**. The first decision we had to make was to choose a programming paradigm that combined these three characteristics.

Object-oriented programming (OOP) has been used since the 1960s in software development and was the default option in the 1990s, the principal paradigm to learn if you wanted to be a good software developer. However, OOP has slightly lost popularity over time, even if in some cases, it is still the obvious choice. The purpose of OOP is to divide services into objects with internal properties, allowing for easy addition or modification of features without impacting the rest of the code.

When applied strictly, developers realize that OOP suffers from numerous problems, particularly in **real-time** and **IoT development**. Indeed, it can lead to hard-to-read code and increase the debugging and maintenance time, with a deep hierarchy due to inheritance and abstraction properties. Multiple abstraction layers can lead to overhead in limited resource environments, due to the need to manage multiple objects. **Mutable state objects** is a significant problem in **real-time processing**, complicating concurrency; in **multithreading**, for example, it can lead to data leaks and data access issues.

In Floware Vision (particularly the **computer vision** part), we didn't find any use cases that would require a mutable internal state. These are the main reasons for our choice not to use OOP in the main part of Floware Vision. Additionally, how do you define objects objectively? What would happen if an object fits into multiple categories? (See The "Platypus" Effect[28]).These questions are not necessary and would make Floware lose time and money.

Another programming paradigm that has gained popularity in the last decade is **functional programming**[29]. While **OOP** focuses on interacting or communicating with objects, in **functional programming** the emphasis is on transforming them[28]. The focus in functional programming is to pass an object, data, or variables into a function that returns a new object, data, or variables representing the transformed input, without altering some internal state in any way. This is achieved using **immutable objects** and **"pure" functions** as described previously.

This lack of a mutable internal state is recognized as a great way to achieve **parallel computation** because it avoids concurrency and accessibility issues. It makes the code more readable and debugging easier. **Pure functions** also significantly improve reliability in data streams because they ensure consistent output for the same input. Furthermore, the code is less bloated with complex hierarchies and inheritance, resulting in shorter, less complex code that enhances maintainability. This simplicity and the reduced failure that induced is are researched in IoT programming [30]. Our sensors needed a paradigm that fitted best our constraint. And **functional programming** is this paradigm.

8.1 Implementation Choice

We noticed quickly that Floware Vision is an aggregate of functionalities that didn't communicate; the computer vision script does not interact with the Bluetooth script, and the sender system gets the data from the folder directly. It is the same for the power management and disk usage services. The application was already composite because each "service" was launched independently, but it wasn't implicit.

Besides the functional programming choice as a programming paradigm, a microservices[4] architecture was chosen. By its composite structure, it facilitates the debugging and updating of each service, but also the deployment. Indeed, teamwork is improved as each member of the team can access a service and upgrade it independently. Services are faster to build, test, and

[4]Microservices - also known as the microservice architecture - is an architectural style that structures an application as a collection of services[31]

deploy[5].

Even though functional programming is the main choice, we don't exclude the use of OOP or procedural programming when it is the most appropriate choice. For instance, some services may need different implementations (conversion purposes or sending to different servers in different formats using different libraries). In this case, using a Strategy Pattern[32] in OOP can be useful, or we can also functionally translate this pattern, using custom-typed functions as parameters in another function.

8.2 Improvement On The Existing

After the bug fixes, we started by refactoring every Python script of Floware Vision to translate procedural computation into a corpus of functions and integrate a configuration JSON file for each script, replacing hardcoded information and mutable object attributes in certain scripts like Analytics. We also removed all the unused files. And organized the file hierarchy by type (models, configuration files, scripts, and services).

Here are some of the improvements in the Python version of the application:

- Added Python documentation to all functions and a general README.
- Removed unused files.
- Refactored the Sync-folder service to use a JSON config file, allowing for the selection of the desired server.
- Automatically select the video device in the Vision service.
- Bugs Fixes
- Fusion of the jetson nano orin, and dGpu application, it can now run on every device.

Floware is also required to modify the Deepstream pipeline of the computer vision to add more classifiers depending on the specificities of each mission, which is easier to do since the refactoring of the script.

[5]More details in the deployment section

8.3 AI Model Integration

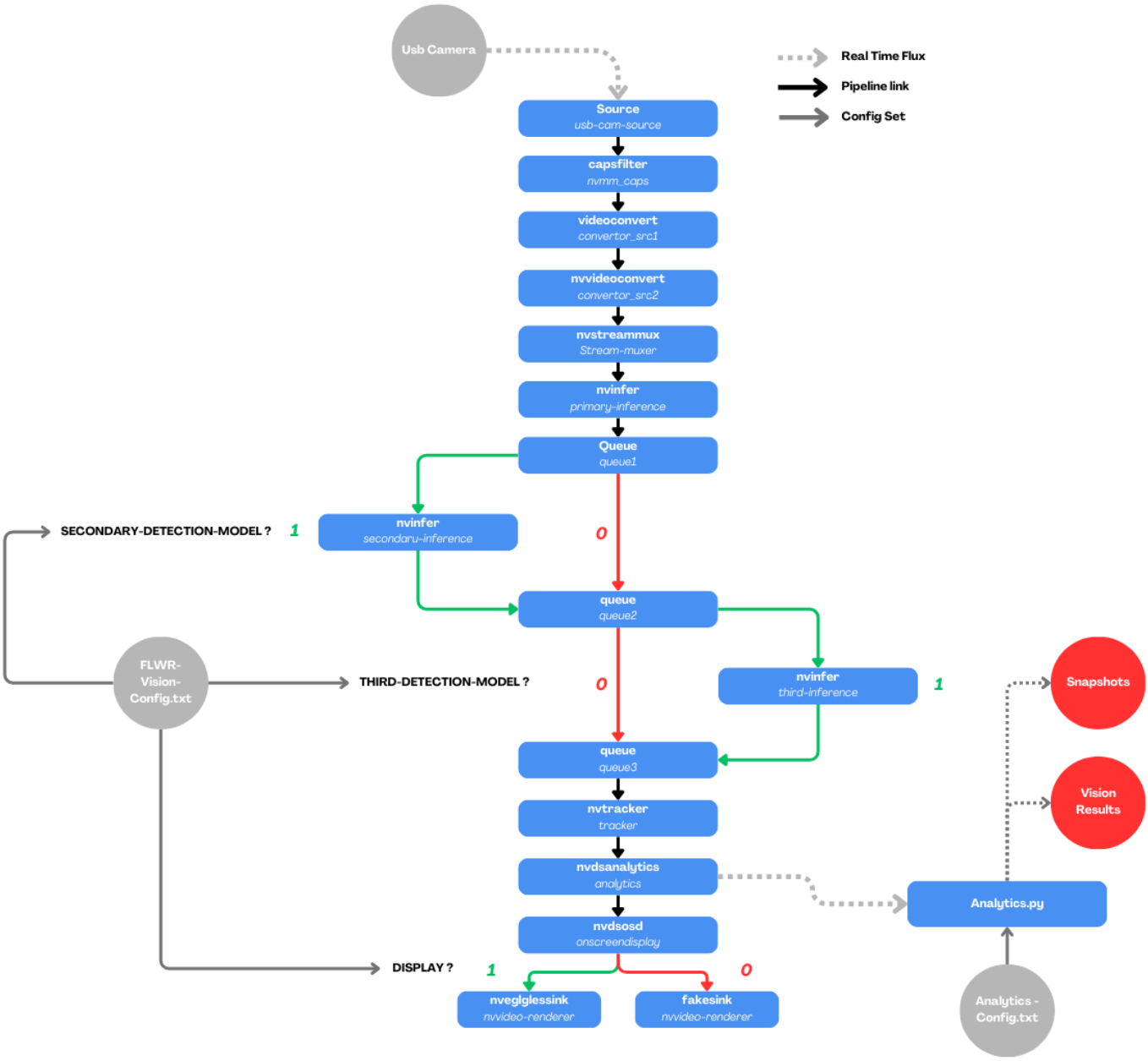


Figure 18: Deepstream Pipeline

Compared to the previous existing pipeline, we can notice the addition of an optional secondary and third classifier. It is currently used for license plate recognition. Using custom Licence plate Detection and text Recognition models provided by Nvidia[33]

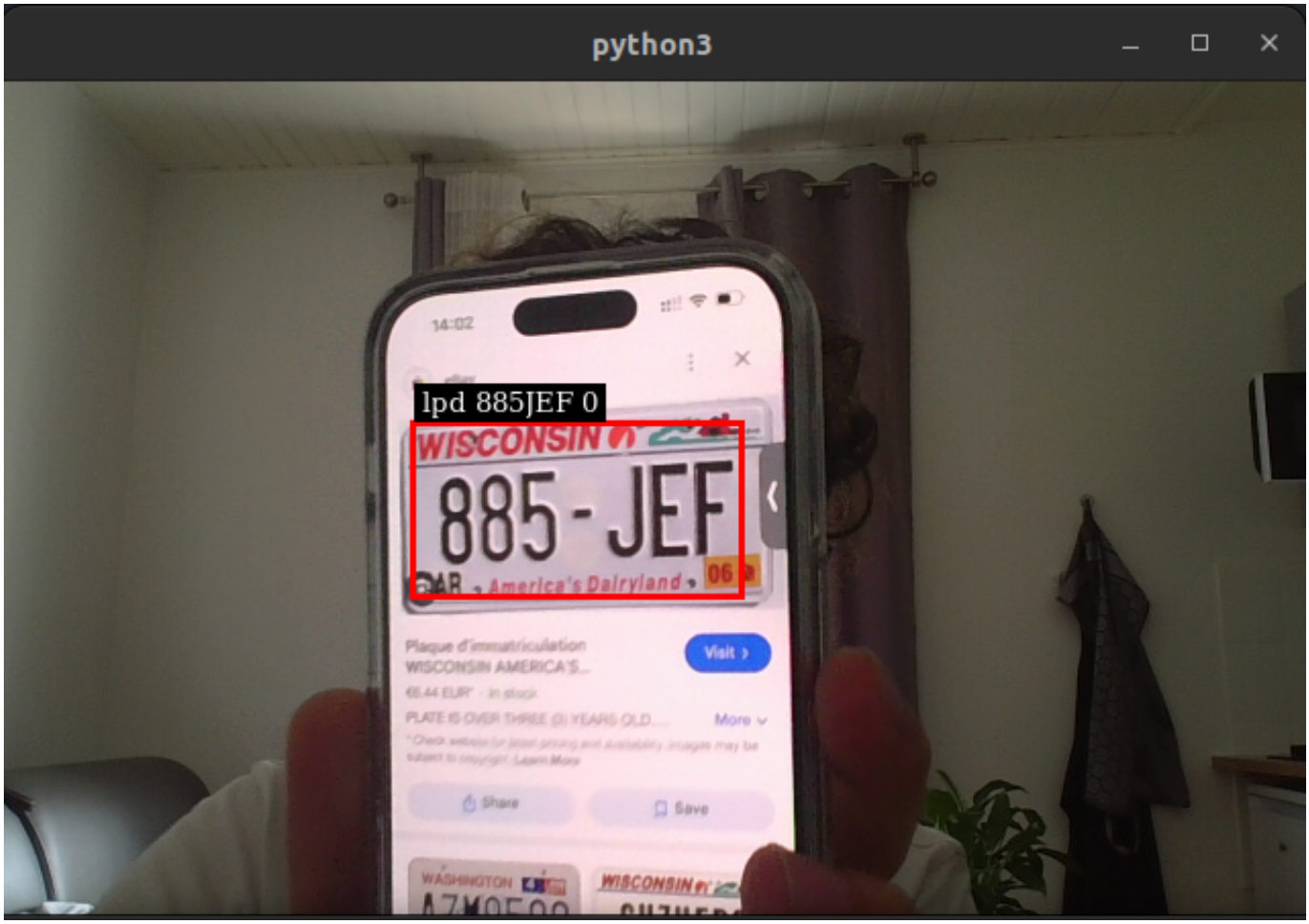


Figure 19: Testing Of Licence Plate Recognition feature

8.4 Floware Vision Bedrock[6]: The C++ Version

We made limited changes to the Python app to ensure reduced deployment and maintenance time, but we knew Python was not the best option for IoT programming where we needed to be closer to the metal[7] in an optimized way. C++ remains one of the most used and fastest languages.

8.4.1 Justification for C++

Nvidia Deepstream and Gstreamer are C++ libraries. By using the Deepstream Python bindings, we are adding an intermediate layer that is not necessary. Using C++ natively improves performance and also simplifies the deployment process and the edge device. Not having to install Python bindings[34] also improves compatibility between devices by avoiding any issues caused by Python or pip, but also reduces the bload of IoT. The fewer elements we have to care about, the fewer issues will happen.

C++ is natively typed and supports both OOP and functional programming[35]. It contains every library needed for our services (even if our architecture permits to use of different languages depending on the service) like Azure Iot, Deepstream, and Ubertooth. We choose the classic C++ naming convention.

[6] A well chosen name.
[7] Closer To The Hardware

8.4.2 Design & Architecture

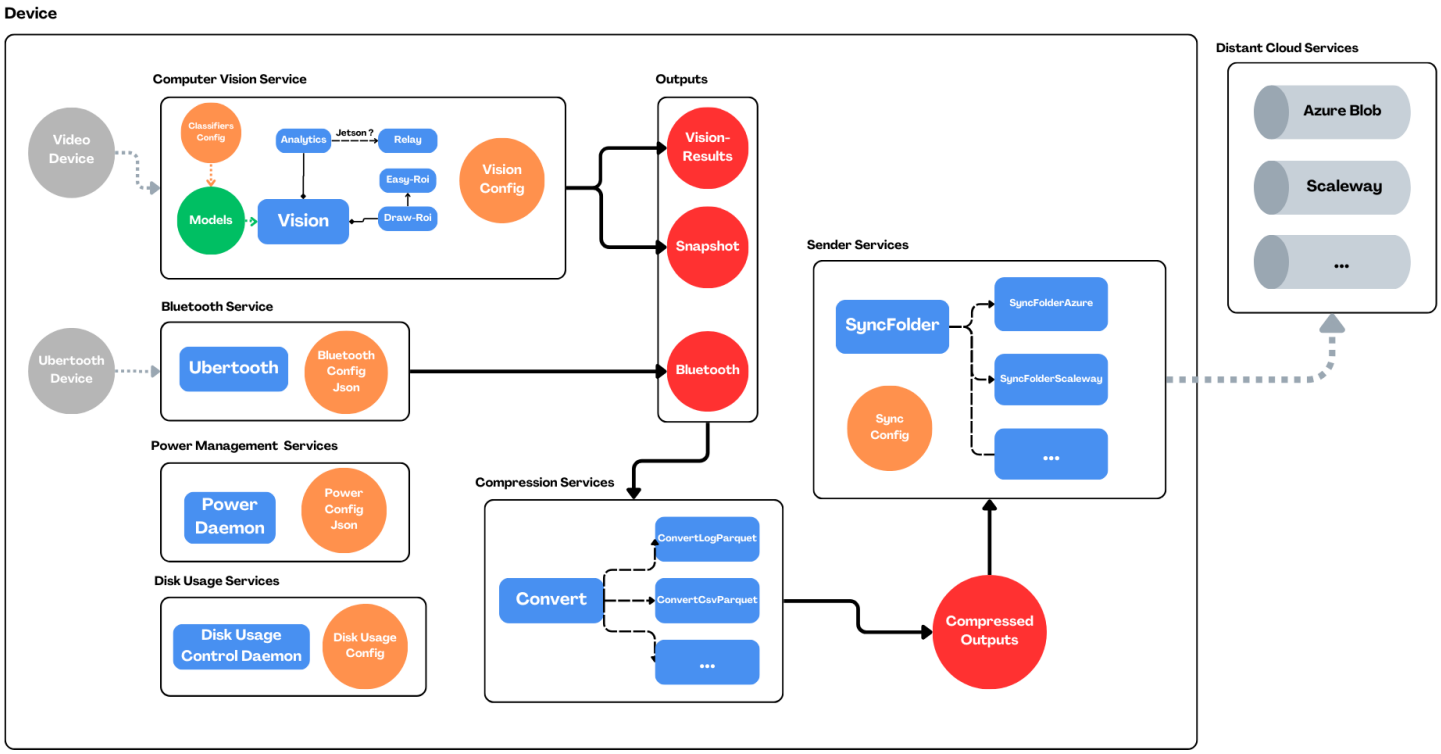


Figure 20: C++ Floware Vision Bedrock Architecture

The architecture graph shows how the data is processed from left to right, providing a clearer view of the microservices architecture. Each service is contained with its libraries or configuration files. Notably, the convert-to-parquet service, which was previously (see 15) implemented in two different scripts—`FLWR-Bluetooth` for Bluetooth data and hardcoded in the `FLWR-Vision Script` for computer vision data—is now an independent service allowing multiple conversion methods. Similarly, the sync folder service has been improved. Floware decided to abandon Virtual Machine storage in favor of using Azure Blob Storage[36] for cost efficiency, management, and scalability. By using the Azure library inside the application, we need to ensure the possibility of multiple implementations to facilitate future migrations to other services. Floware seeks European partners. We could have the need, in the future, to use a European cloud instead of Microsoft Azure.

9 Floware Vision: Deployment

Floware possesses 46 sensors, and deploying and updating them efficiently is crucial for maintaining our reputation with clients. As mentioned above, the initial deployment and debugging method used TeamViewer[37] file transfer system. This method was not scalable because it was not designed for this purpose. Deploying 10 sensors required two people to spend one and a half days. Even before refactoring the legacy code, our main challenge was to find scalable and efficient ways to deploy and update Floware Vision.

9.1 Deployment Strategies

Setting up a Jetson from scratch can be time-consuming due to the number of dependencies to install, such as Gstreamer, Deepstream, and Python bindings. Noticing the significant time loss during this initial setup, we decided to use a container system. This approach allows us to include all dependencies inside a container and then run it to execute any application stored inside a pre-setup environment. We chose to use Docker containers[38] for this purpose.

Where to store the created container? Many solutions exist, like Azure Container Registry, GitHub Container Registry, or

DockerHub. We chose Azure Container Registry for its capacity (GitHub Container Registry is restricted to 500MB per container) and for practicality, as we already have a startup Azure Subscription.

By modifying the Dockerfile[8] and refactoring the application, we reduced the container image size from 16 gigabytes to 3.4 gigabytes.

With this container, we have an object we can easily pull and run. The next challenge was scalability: how to automate the deployment process?

9.2 Introduction to Azure IoT

According to Microsoft: "IoT Hub is a **managed cloud-hosted service** that acts as a **central message hub** for **bi-directional communication** between your IoT solution and the devices it manages." [39]. Azure's key feature is **Device Management**; it simplifies the process of adding new devices to the Azure IoT Hub and also allows the **monitoring** of their status or performance while **updating device firmware** over the air. Our main issue with TeamViewer was **scalability**; Azure IoT can manage hundreds of devices simultaneously. Additionally, the **security** of our devices is improved by having **unique credentials per device** and managing **permissions for each device group** with custom user creation.

Each device can then be recorded as an Azure Edge module and deployed with our Docker images directly from our Azure Portal or Visual Studio Code.

9.3 CI/CD: Actual Deployment Pipeline

If we use Git, a good deployment pipeline also means a good versioning pipeline. We chose to create a Floware GitHub repository to store every project the company has. We defined strict conventions to ensure the most secure deployment process. First, we defined the production branch—the one that contains the version of the code we deploy. We protect this branch to avoid any arbitrary other branch merging or committing without a review. If we want to add a modification to the code, we must create a new branch with a name reflecting its purpose (e.g., `fix/ubertooth`, `feature/parquetconversion`, `update/syncfolder`).

When the fix or feature update is implemented, the user should create a pull request[9] into a development branch. The senior developers then review the modifications and decide whether to merge them. When a new version of the code is finished in the development branch, the content is transmitted to a stash branch for testing and debugging purposes, so developers can still add new modifications to the development branch.

When every feature is tested, we merge the code into production. We can now create a GitHub Release from this production branch with a version tag, representing the version of the application (for instance, flwr vision v1.2.3).

[8] The file that contains the build instruction of the docker image
[9] A request to add these modifications into another branch

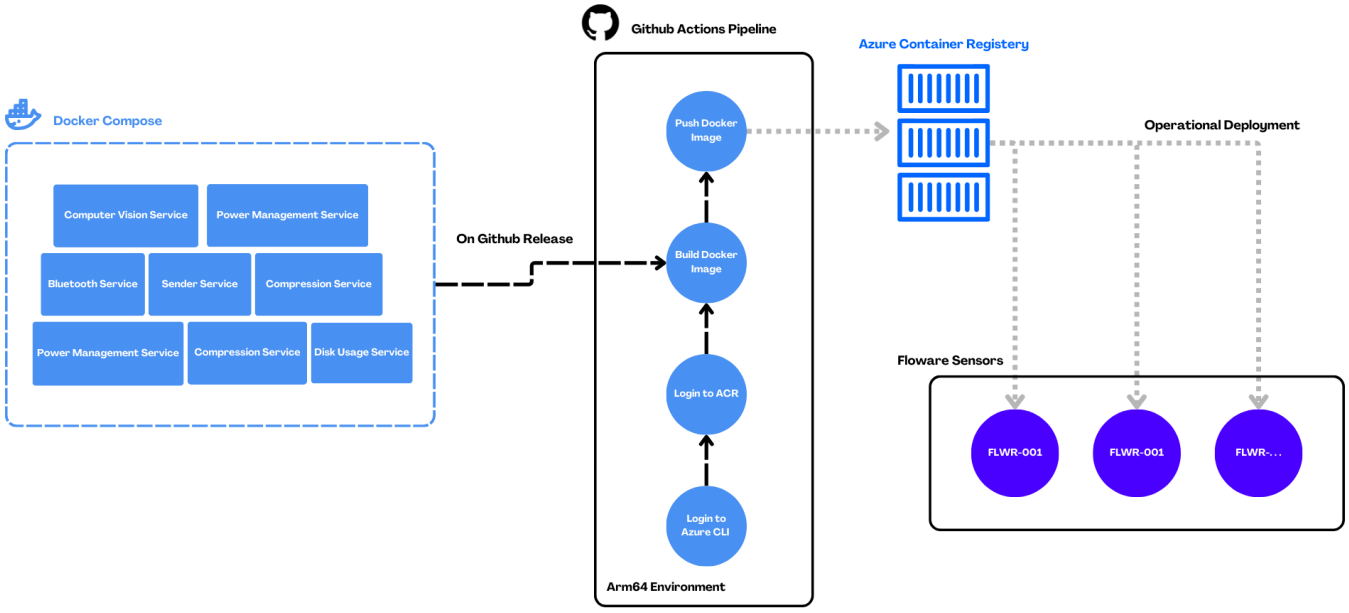


Figure 21: Deployment Pipeline representation

After this release, we create a GitHub Action pipeline triggered at every new release that takes place in a Linux arm64 environment (same as a Jetson) that contains a list of automated instructions. Here, the GitHub Action connects to Azure Container Registry with secretly stored login credentials[10]. The Docker image is then built and pushed to ACR with the version as a container tag. We can then deploy the images to all or a fraction of Azure IoT Edge devices and receive metrics and status updates to ensure the deployment goes right.

10 Results and Discussion

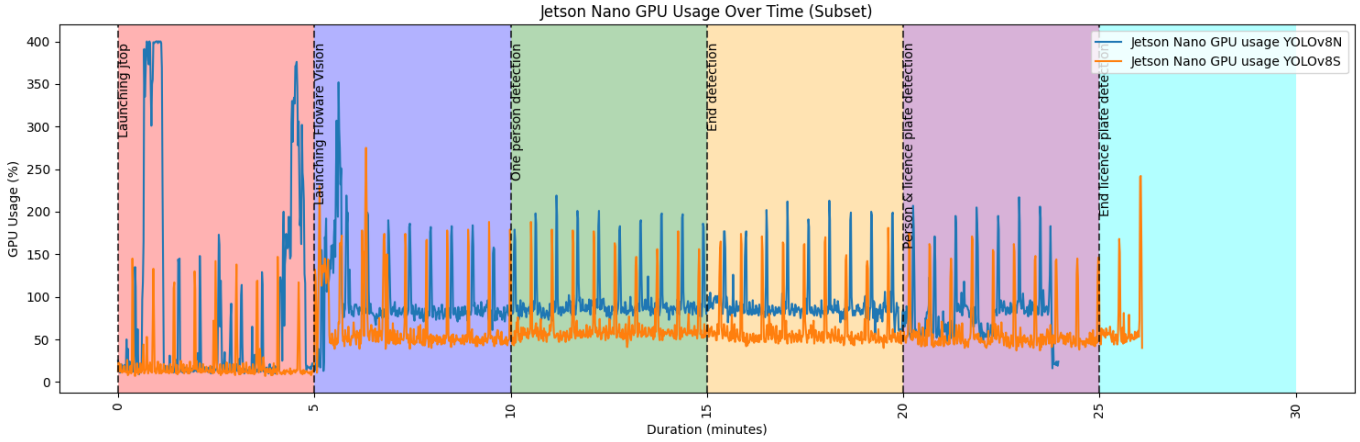
10.1 Floware Vision Performance Metrics

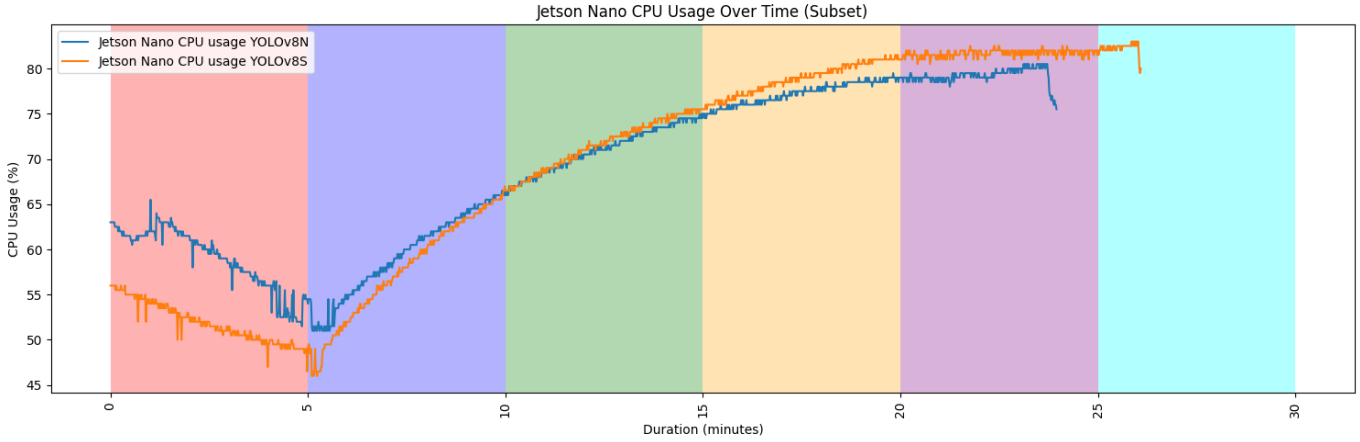
Floware Vision can use two performance metrics sender, deepstream perf for the framerate of floware vision, and jtop, a software that extracts every property of a jetson in real-time, it can be useful for the creation of failure prediction models, or create performance benchmarks on the different Floware vision version.

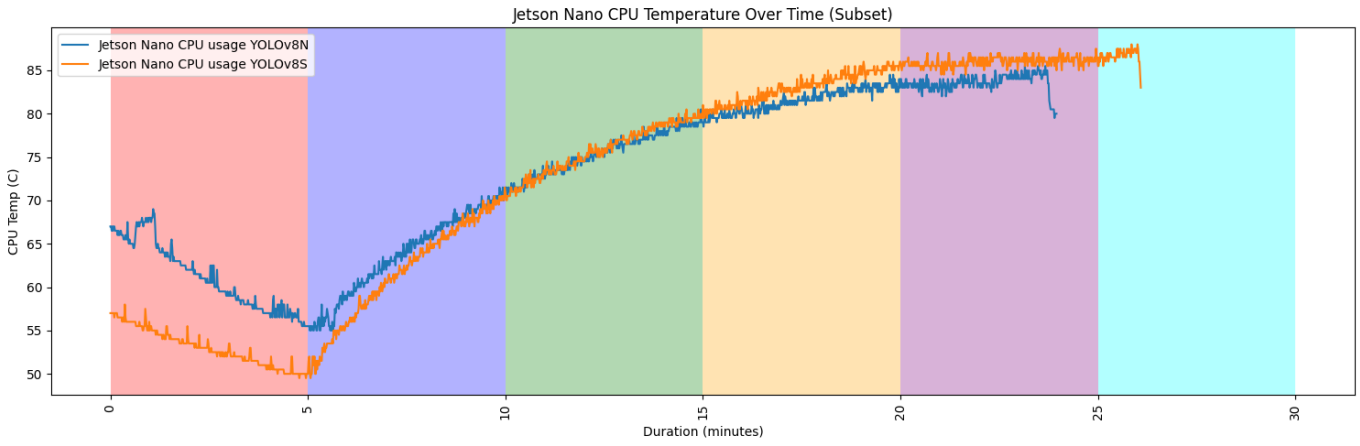
10.2 Case Studies : Yolo8s vs Yolo8n performance

YOLOv8s[41] and YOLOv8n are designed for different computational capacities and use cases. YOLOv8s is optimized for a balance between speed and accuracy, making it suitable for applications where moderate hardware resources are available. YOLOv8n, on the other hand, is designed for scenarios where minimal computational resources are required, prioritizing speed and efficiency. Having a latency issue on Floware Vision, it's important to know which model has the best accuracy for resources taken.

[10]See GitHub Secrets[40]







- **Launching Floware Vision**
  - **YOLOv8s FPS**: 8.8 - 9.2
  - **YOLOv8n FPS**: 17.2 - 17.9
- **One person detection**
  - **YOLOv8s FPS**: 7.6 - 8.4
  - **YOLOv8n FPS**: 13.0 - 15.6

- **End detection**
  - **YOLOv8s FPS**: 7.6 - 8.4
  - **YOLOv8n FPS**: 13.0 - 15.6
- **Person & license plate detection**
  - **YOLOv8s FPS**: 5.7 - 8.8
  - **YOLOv8n FPS**: 11.0 - 15.3

Despite greater GPU usage, we observe better framerate on YOLOv8n and reduced CPU usage, indicated by a flatter courb compared to YOLOv8s. We still need post-process data analysis to judge if the model size loss justifies the better performance, as framerate can affect tracking accuracy.

10.3 Output Analysis: the end side of the pipeline

10.3.1 Visualization of Flows

The first task is visualization. We aggregate sensor data for one or two days using the panda's library. We determine the central point of the bounding box, then for each object, we draw a line between each point. The challenge lies in reducing the noise points. The linear interpolation method was then used to obtain clean and viewable trajectories.

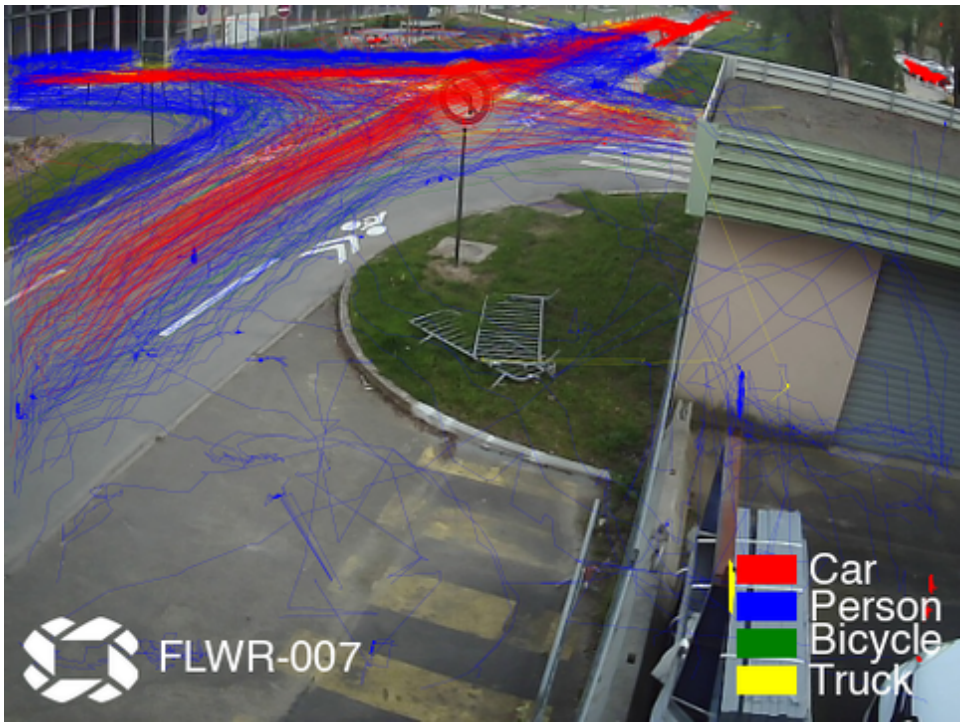


Figure 22: Flux Visualization of sensor FLWR-007

10.3.2 Transition Matrix

This task was carried out in two parts. First, we need to determine regions of interest (ROIs) for each sensor (there are 15 sensors, so some task automation is required). Then, for a chosen time step, we determine the number of movements from one ROI to another for each vehicle class. The required output is a matrix in the following form:

interval	transition	nb_transitions	mode	capter_name	verification
16/09/23 07:00-07:15	T1-2	0	car	FLWR-004	OK
16/09/23 07:00-07:15	T1-3	5	car	FLWR-004	OK
16/09/23 07:15-07:30	T..-..	...	...	FLWR-00...	OK

10.3.3 Activity Diagram

The required output is a diagram in the form of a CSV document usable for PowerBi, representing the activity time of a sensor over a day, week, or month. The sensor is considered inactive if there is no data for an extended period.

10.3.4 Post-Alerting

We check the files received on the server side, and if any data is missing, we send a Slack alert to address the issue as quickly as possible in the following format: `[Mission][Importance Level][Sensor Name][Data Type (Bluetooth or Computer Vision] No DATATPE file received since N days, TIME`

Example of an alert output:

```
[ECOFLOW][RED][FLWR-017][BT] No BT file received since 14 days, 19:36:31.251712
[ECOFLOW][RED][FLWR-017][CV] No CV file received since 14 days, 19:38:30.996416
[ECOFLOW][RED][FLWR-021][BT] No BT file received since 100 days, 13:46:31.340826
[ECOFLOW][RED][FLWR-021][CV] No CV file received since 100 days, 13:02:31.331053
[ECOFLOW][RED][FLWR-022][BT] No BT file received since 21 days, 2:07:31.396548
[ECOFLOW][RED][FLWR-022][CV] No CV file received since 21 days, 2:05:31.379133
```

10.3.5 Queue Length Measurement

For each ROI, we aim to determine the average distance thresholds traveled by each vehicle to classify them as moving, slowing down, or stopping. Then, we choose a distance for each ROI that corresponds to the queue length and their states over time based on the average mobility states of all vehicles. By cumulating the queue lengths for several ROIs according to the direction of traffic, we obtain a queue length for each congestion period, presented in a table format with columns for time and queue length.

10.3.6 Projection

The problem with data analysis on our sensors is that distances are relative to the orientation of the sensor or the road, meaning the distances between points do not correspond to real distances. We project the points from the sensor image onto the satellite image using various research methods, including the RANSAC regression algorithm[42].

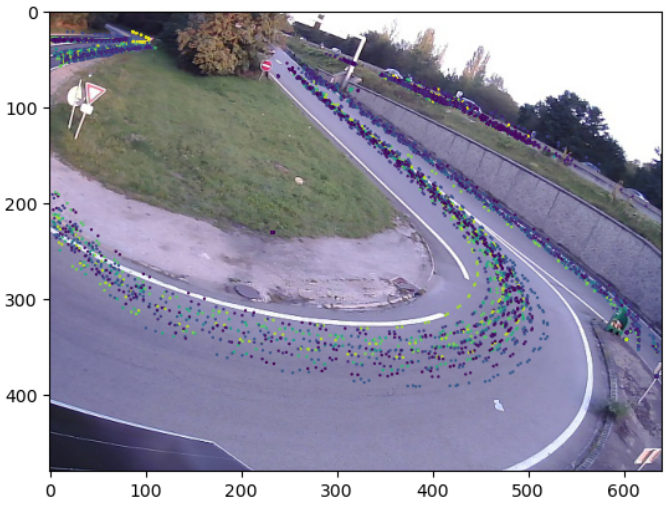


Figure 23: Scatter Plot of Detections on Sensor Images

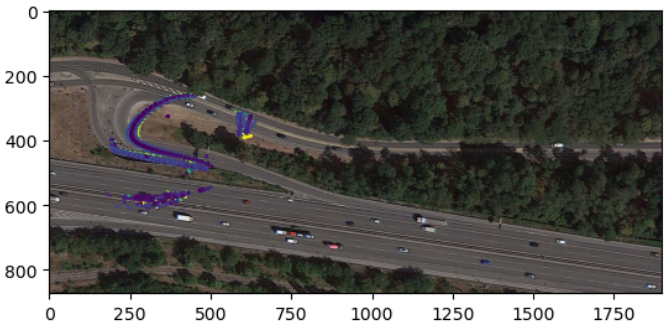


Figure 24: Scatter Plot Projected onto a Satellite Image of the Sensor Location, Still In Testing Stage

11 Conclusion

Writing this memoir made me realize the importance and advantages of Floware's choice to use Edge AI for traffic analysis, due to its scalability, data privacy in public spaces, and real-time processing capabilities. Additionally, it highlighted the need for a robust and maintainable architecture and deployment process to ensure trust and reliability for our clients. The future work will focus on completing our deployment plan in parallel with finishing the development of Floware Vision Bedrock. After that, the next few months will be dedicated to training and testing new computer vision models fine-tuned for specific mission tasks. For instance, as part of our partnership with Vinci, we need to fine-tune a YOLO model for classifying various types of trucks. In the long term, Edge AI will not be the only AI paradigm used. Work on flux simulation model and large language model for interrogating our database. I am confident that the continued development and refinement of our Edge AI solutions will contribute significantly to advancements in traffic management and urban mobility. We are confident that our ongoing efforts to develop and refine our Edge AI solutions will play a significant role in advancing traffic management and urban mobility.

References

[1] "Modules et kits de développement pour systèmes Embedded | NVIDIA Jetson." https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/.

[2] M. Treiber, A. Kesting, and C. Thiemann, "How Much does Traffic Congestion Increase Fuel Consumption and Emissions? Applying a Fuel Consumption Model to the NGSIM Trajectory Data,"

[3] "Institut vedecom website." available at https://www.vedecom.fr/accueil-2/ite/quisommesnous.

[4] "Moove Lab – L'accélérateur des startups de la mobilité de Station F." available at https://moove-lab.com/en/.

[5] "Leonard, la plate-forme de prospective et d'innovation de VINCI." https://leonard.vinci.com/.

[6] "Nextmove : Pôle de compétitivité Européen de la mobilité." https://nextmove.fr/.

[7] "Ecomesure." https://ecomesure.com/en.

[8] "FabMob France." https://lafabriquedesmobilites.fr.

[9] "Wintics • Cityvision, video analysis software for urban stakeholders." https://wintics.com/en/.

[10] "Collecte & valorisation des données de mobilité." https://www.alyce.fr/.

[11] "Accueil, UPcity." https://www.up-city.be/.

[12] "Eurovia." https://www.eurovia.fr/.

[13] "Accueil | CDVIA." https://www.cdvia.fr/.

[14] S. Denning, "What Is Agile?." https://www.forbes.com/sites/stevedenning/2016/08/13/what-is-agile/.

[15] "Cloud Computing, sécurité et réseau de diffusion de contenu (CDN)." https://www.akamai.com/fr.

[16] "What Is Edge AI? | IBM." https://www.ibm.com/topics/edge-ai, Aug. 2023.

[17] K. Chowdhary, "Microsoft Azure Hit With The Largest Data Breach In Its History; Hundreds Of Executive Accounts Compromised." https://techreport.com/news/microsoft-azure-hit-with-the-largest-data-breach-in-its-history-hundreds-of-executive-accounts-compromised/, Feb. 2024.

[18] "Prospera.ag." https://prospera.ag.

[19] "Site officiel Fitbit : coachs électroniques pour la forme et le sport, et bien plus encore." https://www.fitbit.com/global/fr/home.

[20] "What Is Computer Vision? | Microsoft Azure." https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-computer-vision.

[21] K. Fukushima, "Neocognitron," *Scholarpedia*, vol. 2, p. 1717, Jan. 2007.

[22] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," May 2016.

[23] S. Sánchez Hernández, H. Romero, and A. Morales, "A review: Comparison of performance metrics of pretrained models for object detection using the tensorflow framework," *IOP Conference Series: Materials Science and Engineering*, vol. 844, p. 012024, 06 2020.

[24] "Ubertooth One - Great Scott Gadgets." https://greatscottgadgets.com/ubertoothone/.

[25] "GStreamer/gstreamer." GStreamer GitHub mirrors, July 2024.

[26] "DeepStream SDK." https://developer.nvidia.com/deepstream-sdk.

[27] "Ultralytics: Ultralytics YOLOv8 for SOTA object detection, multi-object tracking, instance segmentation, pose estimation and image classification.."

[28] Talin, "The Rise and Fall of Object Oriented Programming," Nov. 2018.

[29] S. Akhmechet, "Functional Programming For The Rest of Us." https://www.defmacro.org/2006/06/19/fp.html, June 2006.

[30] T. Hänisch, "A Case Study on Using Functional Programming for Internet of Things Applications," *Athens Journal of Technology & Engineering*, vol. 3, pp. 29–38, Feb. 2016.

[31] "What are microservices?." http://microservices.io/index.html.

[32] V. Thennakoon and B. Hettige, *A STUDY ON OBJECT-ORIENTED DESIGN PRINCIPLES AND PATTERNS*. July 2022.

[33] "NVIDIA-AI-IOT/deepstream_lpr_app." NVIDIA AI IOT, June 2024.

[34] "NVIDIA-AI-IOT/deepstream_python_apps." NVIDIA AI IOT, July 2024.

[35] kexugit, "C++ - Functional-Style Programming in C++." https://learn.microsoft.com/en-us/archive/msdn-magazine/2012/august/c-functional-style-programming-in-c, Jan. 2016.

[36] "Azure Blob Storage | Microsoft Azure." https://azure.microsoft.com/en-us/products/storage/blobs.

[37] "TeamViewer - Le logiciel de connectivité à distance." https://www.teamviewer.com/fr/.

[38] "Overview of the Docker workshop." https://docs.docker.com/guides/workshop/, 11:50:25 -0700 -0700.

[39] leestott, "Introduction - Training." https://learn.microsoft.com/fr-fr/training/modules/introduction-to-iot-hub/1-introduction.

[40] "Using secrets in GitHub Actions." https://docs.github.com/_next/data/cjISr1MlIQhzdaVAuo8xo/en/free-pro-team@latest/actions/security-guides/using-secrets-in-github-actions.json?versionId=free-pro-team%40latest&productId=actions&restPage=security-guides&restPage=using-secrets-in-github-actions.

[41] Ultralytics, "YOLOv8." https://docs.ultralytics.com/models/yolov8.

[42] M. Rezaei, M. Azarmi, and F. M. P. Mir, "3d-net: Monocular 3d object recognition for traffic monitoring," *Expert Systems with Applications*, vol. 227, p. 120253, 2023.