UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**PROIECT DE DIPLOMĂ**

Absolvent:

Menchiu Bogdan-Alexandru

Specializarea Embedded Systems

–Sibiu, 2024–

UNIVERSITATEA “LUCIAN BLAGA” DIN SIBIU

FACULTATEA DE INGINERIE

DEPARTAMENTUL DE CALCULATOARE ŞI INGINERIE ELECTRICĂ

**PROIECT DE DIPLOMĂ**

Tracking objects using deepstream through multiple streams

Conducator ştiinţific: Prof. dr. ing. Florea Adrian

Absolvent:

Menchiu Bogdan-Alexandru

Specializarea Embedded Systems

# Cuprins

[Cuprins 3](#_Toc165638532)

[1 Introduction 3](#_Toc165638533)

[1.1 Prezentarea temei 3](#_Toc165638534)

[1.2 Motivation for choosing the topic 4](#_Toc165638535)

[1.3 Utilitate 4](#_Toc165638536)

[2 Theoretical consideration 6](#_Toc165638537)

[2.1 Technologies used 6](#_Toc165638538)

[2.1.1 Python 7](#_Toc165638539)

[2.1.2 Jetpack SDK 8](#_Toc165638540)

[2.1.3 Nvidia Jetson Nano A02 11](#_Toc165638541)

[3 Building Real Time Video AI Application 13](#_Toc165638542)

[3.1 Proiectarea sistemului 13](#_Toc165638543)

[3.1.1 Analyses and specification of requirments 13](#_Toc165638544)

[3.2 Arhitectura aplicației 14](#_Toc165638545)

[3.2.1 The General Use Case of the Application 15](#_Toc165638546)

[3.3 Set up the environment 16](#_Toc165638547)

[3.3.1 Set up Jetson Nano board 16](#_Toc165638548)

[3.3.2 Configure Development Environment 17](#_Toc165638549)

[3.4 Development of the application 18](#_Toc165638550)

[3.4.1 Pipeline Components 18](#_Toc165638551)

[3.4.2 Initializing Pipeline 19](#_Toc165638552)

[3.4.3 Creating pipeline components 20](#_Toc165638553)

[3.4.4 Linking and Adding to pipelin 21](#_Toc165638554)

[3.4.5 Starting the pipeline 22](#_Toc165638555)

[3.5 Scalabilitate și performanță 23](#_Toc165638556)

[4 Concluzii 24](#_Toc165638557)

[5 Bibliografie 24](#_Toc165638558)

# Introduction

## Prezentarea temei

In the last decade, technology has advanced in every imaginable field. From nanotechnology to remotely controlled chips, technology has expanded and changed at a pace faster than ever before. The evolution of hardware components occurs in parallel with the evolution of software tools to meet their demands. Information technology has been advancing exponentially lately, and we are becoming increasingly dependent on it. Nowadays, we are surrounded by smart devices that make our lives easier.

This project exploits NVIDIA's DeepStream SDK to build an object-class throw multiple video streams in real time. Efficient utilization of GPU-based computing and deep learning techniques is a critical part of a plan to process video streams and verify if there are some objects that are traceable. This increase is predicted to bring a significant improvement in the accuracy of detection tasks that usually suffer from latency throughout those processes. I aim to build a durable system that not only has a proof-of-concept and can be implemented in the field of applications, but also used in automated industrial operations, security surveillance inside a factory.

This project implementation is forecasted to take a step forward in the video analytics area by means of a prompt and accurate data processing that only real-time system of processing can provide.

## Motivation for choosing the topic

The connection of tens of billions of cameras and sensors to the global networks makes the collection of the big amounts of data possible. This makes it not only possible to generate insights for business but also to improve the process efficiency and even to generate revenue opportunities. Highly adaptable, these technologies can be leveraged in different environments for ensuring the flows such as in traffic intersections to decongest traffic, monitoring health and safety incidents in hospitals, providing great customer experience in shopping environments and detecting defects outcoming production. IVA must be highly reliable and the scenerios should be presented 'on the fly' automatically.

What drives to build up the real-time object detection system with the help of NVIDIA’s DeepStream SDK is the rising need for complex monitoring and surveillance systems that is directly rooted in the present day's fast-growing technological environment. The rapid development of real-time object detection is essential for the further development of safe conditions in public spaces, efficient traffic management, and input across multiple sectors like transport, retail, or industry.

In the industrial sector, real-time object detection promotes continuity of production with improved asset identification and management process. Real-time object detection also enables quick inventory control and better robot programming. It is what ensued the reactive monitoring of equipment and machinery becoming possible through timely detection of impending problems. This consequently reduced the period of downtime and the expenses incurred during maintenance.

The application of computation power of NVIDIA GPUs and the sophisticated functionalities of the DeepStream SDK makes it possible to extend the horizon of what is possible today in video analytics. This technique is not only a quick response but also a foundation for future implementations of video analytics in varying environments.

## Utility

Smart video analytics made us rethink the way we interact with and manage our environments which now lead to a huge list of applications that are indeed making a big contributions to various sectors.

Here are some key examples of its transformative capabilities:

* **Access Control:** In sensitive territories like airports, smart video analysis can take security to the next level through careful identification of individuals to allow only authorized people at the entrances, so eliminating the need of manual checks.
* **Managing Operations:** AI Video can accelerate the transportation of logistics and manufacturing channels, providing online real-time decisions for warehouse production and distribution, the option of choosing the best resource allocation and inventory accuracy.
* **Parking Management and Traffic Engineering:** For urban planning and smart city tasks, video analytics become a tool to better manage parking spaces effectively and optimize traffic flow encouraging more volume of people to move around roads safely and reducing congestion.
* **Retail Analytics:** The process of data gathering and analyzing of customer behavior within the stores will be focused on to improve customer experiences, shop layouts, and product display. The aim is to result in more sales, higher customer satisfaction, and profits.
* **Optical Inspection:** In production automatic video inspection systems enable gain precise and consistent checks of assembly lines with defects identification and quality control by-passing the requests of postponing the process not of the production.
* **Content Filtering**: Media platforms apply AI to monitor and standardizing content in order to comply with the rules of the government and the charter of the community as well as to cater to user's wishes by presenting content tailored to their preferences.

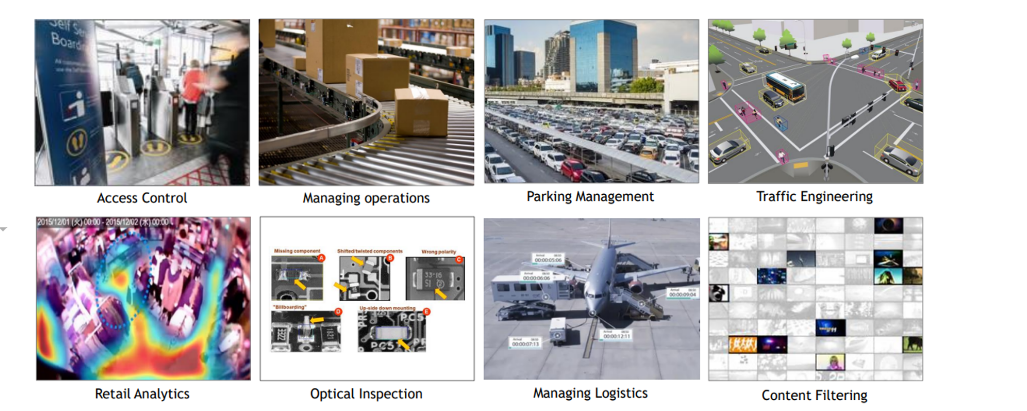


Fig1 Real time video applications[1]

# Theoretical consideration

## Video Application Framework

Almost every video app follows a specific structure with features given in (fig2). Usually, the systems are fed by  one or several video streams piped-in from different sources,  which then are decoded and fused together. Then, these data sources process batches of video sets in order to be ready to be submitted for various processes of analyzing. After this, the generated insights are integrated with the original video footage and can be used in one of three ways: 1)The signal is 2) The information is 3) Data is then 3) The information then is further transmitted downstream for additional analytical processing.

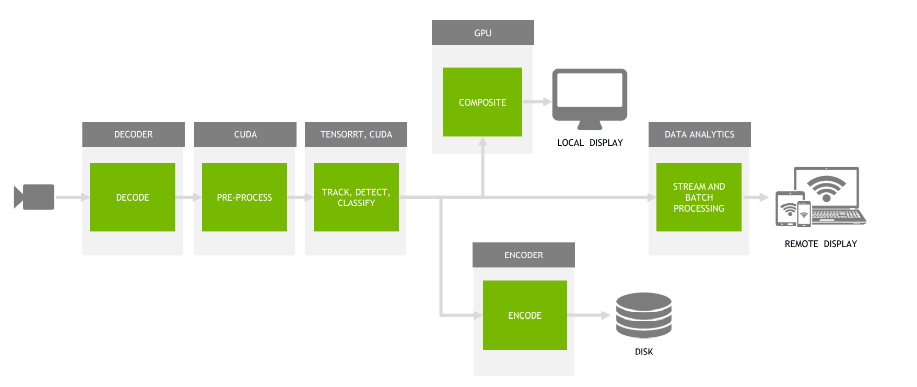
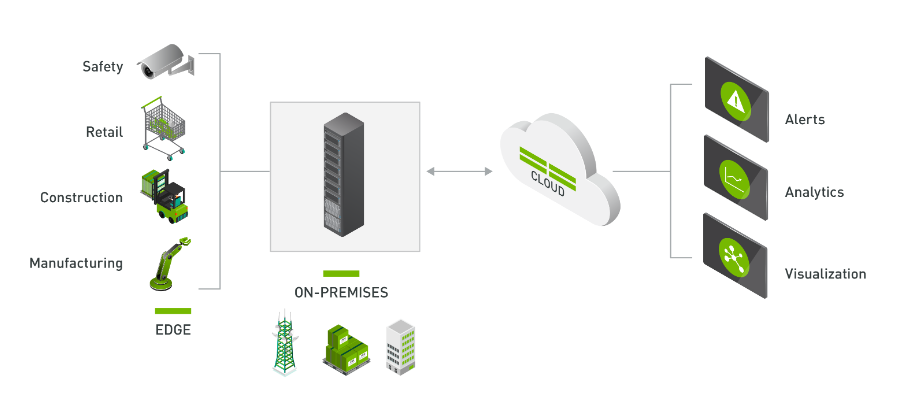


Fig2. Video Application general framework[2]

## Technologies used

Today, the cameras are everywhere, and they metamorphosed into a data capturing agent implementing business analytics, process optimizations, and revenue channels. Converting video content into critical observations is an error-prone and challenging task. The processing can take place at the edge that is near the sensors, on-site or in the cloud. Thus, there is no need for the data transfer from one point to another which is more prone to data loss or alteration. Once the AI generated insights are purged downwards, they can be used for further processing activity, such as generating alerts based on provided criteria, performing further analytics, or even producing different kinds of visualization that show trend and pattern. Designing video Artificial Intelligence applications is a very complicated process since the developers have to develop multifaceted systems, train them (robust neural networks), and comprehend the scope/ impact of the design choices.



As with any other software, in the industry of video applications, there are multiple ways to develop these applications. Examples are numerous, they range from coding languages, frameworks to architecture approaches. These all will differ depending on the unique necessity of the product.

For the application I developed, I used the following software and hardware technologies:

* Python – progamming language (owned by: Microsoft)
* Ubuntu 18.05
* Deepstream Framework (owned by: Nvidia)
* Nvidia Jetson Nano developer kit
* Video Cameras ( HykVisison Cameras)
* Harness ( Wifi cable, SD Card, power cable)

### Development platform

#### Python

Python is a general-purpose programming language that was built by Guido van Rossum in the year of 1991.It is credited with its first release in the same period. It is designed for easy reading and writing, with a concise syntax simplified, and is favoured by everyone, be it a beginner or a professional. Python employs numerous approaches of programming, which includes object-oriented, imperative and functional paradigms. It has memory management as one of its features and is dynamically typed. Such type systems fosters rapid applications creation. Python has a powerful internal library with tools to perform all kinds of data-connected tasks from a simple code to the more complex ones; for example, for data structuring and filtering, math problems, code testing, web instances collection and many others.

The DeepStream framework built by NVIDIA takes advantage of the versatility of Python which is most striking in its application. It unifies the machine learning and data handling systems to give developers an easy way to integrate DeepStream’s features into Python-based systems without revamping them. Python can work with a set of libraries and frameworks to enable the system to do more complex analysis, e.g. such as NumPy and TensorFlow.

#### Ubuntu

Ubuntu is a open-source operating system which is derived from Linux. It is characterized for its ease of use and reliability, Ubuntu enjoys widespread popularity for both personal desktop and server applications worldwide. It is particularly esteemed within the developer community for its extensive software support, broad range of libraries, and its ability to work seamlessly with various hardware configurations.

Taking advantage of the robust hardware platforms provided by NVIDIA as well as the DeepStream SDK, this GPU-assisted real-time object detection implementation is deployed on the Ubuntu operating system that boasts a wide selection of stable and rich functionalities for developers. Thereby, Ubuntu's support for NVIDIA technology makes the fusion of the hardware and software components flow smoothly, creating the conducive environment for the system to attain optimal performance and excellent compatibility. Using Ubuntu as the operating system platform ensures that upgrade is easy, and security patches and open-source toolset access are available by just installing the software, thus safe-guarding the advancement of the video analytics solutions.

### Jetpack SDK

The JetPack SDK is an extensive toolkit provided by NVIDIA for building and deploying AI and robotics applications on NVIDIA Jetson devices. This toolkit includes an array of tools, libraries, APIs, and sample projects tailored to leverage NVIDIA GPU capabilities, enhancing the development of AI-centric applications.

Key elements of the JetPack SDK include:

* **CUDA** - a suite of libraries that boost machine learning and deep learning projects, featuring components like cuDNN for neural network operations and TensorRT for optimized deep learning inference.
* **Computer Vision** - essential tools and APIs for processing images and videos, enabling capabilities such as object recognition, image segmentation, and video analysis.
* **Multimedia Processing**: Libraries and APIs designed for effective handling of video and audio processing, crucial for media-rich applications.
* **Operating System and Tools**: The SDK includes a specialized Linux version for Jetson platforms and various tools for application debugging, profiling, and deployment.

#### Deepstream SDK

DeepStream is a software development kit (SDK) provided by NVIDIA designed for building scalable and high-performance applications that process video and other streaming data. It's primarily used for Intelligent Video Analytics (IVA) applications that require real-time analysis and decision-making, such as surveillance, traffic management, customer behavior analysis, and more.

DeepStream consists of a collection of plugins that simplify the creation of a deep learning video analysis pipeline. Developers can leverage the open-source DeepStream SDK to expedite development, thereby minimizing both time and effort needed for the project. As a streaming analytics toolkit, DeepStream enables the development of intelligent systems that utilize artificial intelligence and deep learning to analyze video content.

DeepStream is the part of NVIDIA's broad product line namely including AI, machine learning and deep learning technology which is intended for training and to push modern applications into industries. NVIDIA GPU enhances and allows the smooth development to end with the high performance and AI-driven video analytics applications.

This toolkit is highly customizable working on both discrete GPU architectures (like Nvidia) and on Chip platforms for Jetson devices. It facilitates the development of sophisticated applications by providing robust support and capabilities such as:It facilitates the development of sophisticated applications by providing robust support and capabilities such as::

* Numerous deeplearning frameworks
* Several models working in tandem.
* Various models combined in series or parallel connection to create an ensemble.
* Customized pre- and post-processing
* Computing at different precisions
* Working with Kubernetes

#### Gstreamer plugin

GStreamer is a flexible media processing framework that is widely employed for various multimedia needs such as audio and video watching, recording, broadcasting, and editing. It's basically a strong utility software for media systems and is found almost everywhere starting with different operating systems like Linux, Windows, macOS and embedded devices. Here’s a detailed breakdown of its key features and capabilities:Here’s a detailed breakdown of its key features and capabilities:

* **Pipeline-based Design:** It is advantageous for GStreamer to take the pipeline scheme, where media data moves through a set of elements affecting it successively. This modularity permits different types of jobs to inhabit the same space and still remain functional.
* **Plugin Architecture:** The software is a great platform which provides a broad range of plugins that enhance its functionality and let it to do the media files conversion, codecs, as well as the other additional operations.
* **Cross-platform:** GStreamer is working on various operating systems that means, it gets pretty tasty for a developer working with different systems.

#### The anatomy of pipeline using Deepstream

The program is using the GStreamer and DeepStream which are based on the plugin architecture. This allows us to interact with the environment which is encapsulated inside these plugins. With one plugin, several tasks could be hoisted to make working with plugins easier for developers. Developers when building a pipeline have a sufficient variety of GStreamer/DeepStream plugins that they can adopt to their needs or even design a custom one. Within a pipeline, each plugin is able to reach each functional unit, such as video decoding/encoding, scaling, or inferencing.

The diagram below exemplifies the pipeline of video analytics application that involves taking in video input and processing it, and finally producing analytical outputs. Switching from the general-purpose register to the specific hardware tanks holds the key to higher performance throughout here. For optimal execution, the logic of plugins could be accelerated and designed to handle the memory as smartly as possible with approaches attached like zero copy between plugins and use of multiple accelerators to deliver utility performance.

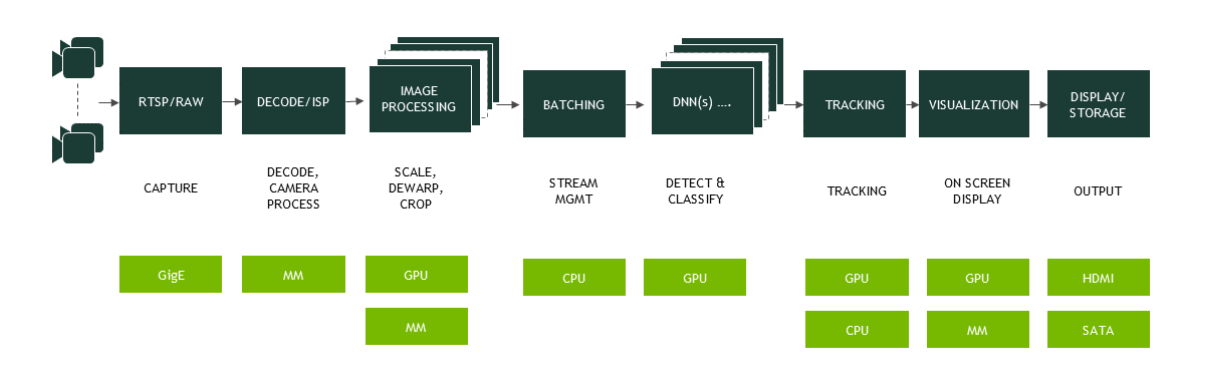


Fig3 The General Arhitecture of a pipeline Deepstream[3]

#### TAO Toolkit

As an AI model adaptation platform, NVIDIA TAO (Train, Adapt and Optimize) Toolkit is absolutely used for simplifying and speeding up the process of creating tailored AI models that can be rapidly deployable for a diverse range of edge products. It is powered by NVIDIA's pre-trained AI models, deep learning expertise making it possible to build your own customizable model by simply adjusting the models for specific applications without going through the complex AI training.

The developed TAO Toolkit is among adventurous models which can be used to accomplish AI activities of vision like detection, classification, and segmentation. This toolkit, which is characterized by a simple and user-friendly design, provides the opportunity to have transfer learning techniques implemented to utilize already pre-trained models, for instance, to meet specific requirements. Besides that, TAO Toolkit equipment features model optimization as one of its tools through model pruning to reduce the model’s size and quantization to improve its performance.

#### CUDA

CUDA, popularly known as Compute Unified Device Architecture by NVIDIA, is a parallel computing platform and an API (application programming interface) which runs side-by-side with the host CPU. It is possible for the software developers and engineers who don’t have the GPUs in the first place to adopt CUDA technology to enable the utilization of a CUDA-compatible GPU for general purpose tasks, which is called GPGPU (General-Purpose Computing on Graphics Processing Units). CUDA allows for close access to the virtual instruction set that is operated by the GPU so it can process instructions in more efficient parallel processing schemes.

It effectively improves the computer applications and exceeds the CPU power when using GPU through CUDA. This advancement enables the speed of advanced applications by dispatching the most demanding sectors of the code to the GPU while the other part process them on the CPU. The fields of application that can take advantage of CUDA are broad in such a way that they find domination in detailed stimulation and analytical computations that are typical in scientific computing and data analysis, as well as real-time 3D graphics rendering and video processing.

### Nvidia Jetson Nano A02

Jetson Nano is the main competence to collaborate with the JetPack and other frameworks such as DeepStream. These system frameworks which are optimized to run smoothly on NVIDIA’s hardware can be used to improve video analytics and robotics among other applications

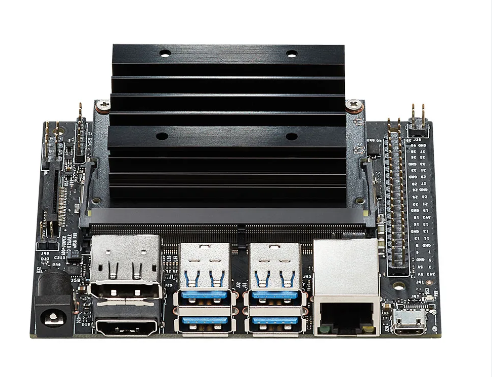


Fig4 Nvidia Jetson Nano[4]

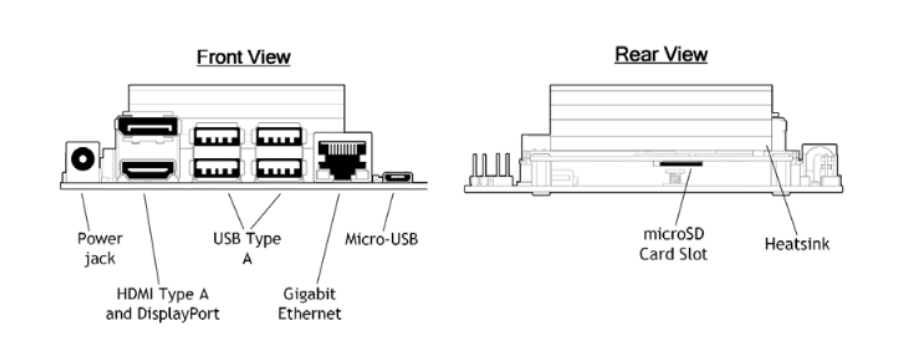
The Jetson Nano from NVIDIA is a tiny and powerful computer that is engineered to run various AI and ML (Machine Learning) tasks. It is an absolutely perfect instrument for development teams, IT scientists and technologists that require AI based devices and solutions.

Fig5 Front and rear view of Nvidia Jetson Nano

The front view shows the device's various connectivity options: a power jack for energy supply, multiple USB Type A ports for peripherals, a Micro-USB port for alternative power or data, HDMI and DisplayPort for video outputs, and a Gigabit Ethernet port for high-speed network connectivity. The rear view highlights the microSD card slot, which is used for storage expansion and loading the operating system, and a heatsink, crucial for dissipating heat generated during intense processing tasks.

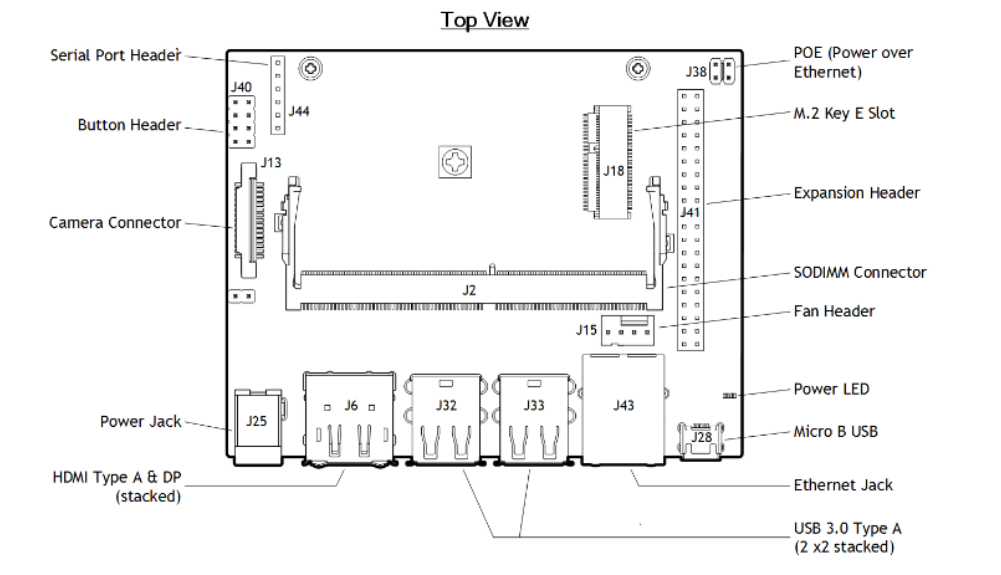


Fig6 Top view of Nvidia Jetson Nano

The Nvidia Jetson Nano board features various connectors and headers for interfacing with peripherals and expansion modules, crucial for developers working on AI and IoT projects. The key components included are described in the next tabel.

|  |  |
| --- | --- |
| **Component** |  |
| Serial Port Header and Button Header | advanced control such as resetting or interfacing with custom hardware |
| Camera Connector | connection of camera modules directly to the board |
| Power Jack | Provides power to the board |
| HDMI and DisplayPort | offers flexible video output options to monitors |
| Power over Ethernet (PoE) | network connectivity |
| M.2 Key E Slo | additional expansion capabilities like extra storage or wireless modules |
| Expansion Header | provides GPIOs for attaching other hardware components |
| SODIMM Connector | used for adding RAM |
| Fan Header | to connect a fan for additional cooling |
| Micro B USB, Ethernet Jack, and USB 3.0 Type A | offer standard connections for data transfer and network access. |

It offers robust GPU capabilities, enabling it to handle complex AI tasks and machine learning operations efficiently. This makes it particularly advantageous for projects requiring real-time data processing and edge computing. Additionally, its array of connectivity options, including USB ports, HDMI, and camera interfaces, ensures versatility, allowing users to easily expand and adapt their setups as needed. The affordability of the Jetson Nano further enhances its appeal, providing a cost-effective solution.

# Building Real Time Video Application

## Proiectarea sistemului

### Analyses and specification of requirments

The application implemented is based on the simultaneous analysis of multiple video streams, as well as the analysis, detection, classification, and tracking of objects across several video cameras that doesn’t overlap. This comprehensive approach enables robust monitoring and surveillance capabilities, ensuring accurate and efficient real-time object recognition and behavior analysis.

In designing the system and defining specific functionalities of the application, the following requirements are established:

* Implementation of an embedded hardware system using the Nvidia Jetson Nano A02, equipped with video cameras
* Integration of JetPack SDK into Nvidia Jetson Nano for video analysis: This requirement focuses on installing and configuring the JetPack SDK on the Nvidia Jetson Nano platform. JetPack SDK is essential as it includes all the necessary drivers, libraries, and APIs that enable the Jetson Nano to perform complex video analysis tasks efficiently
* Integration of DeepStream, TensorRT, and TAO Toolkit libraries and frameworks for analysis and classification: This requirement entails setting up and configuring the DeepStream SDK, TensorRT, and TAO Toolkit on the Nvidia Jetson Nano
* Development of a feature for detecting objects and individuals: This requirement involves creating functionality within the application that enables the precise detection of both objects and people within the video streams processed by the Nvidia Jetson Nano.
* Capability to track objects across multiple cameras and provide real-time positioning: This requirement calls for the development of a multi-camera tracking system that can seamlessly follow objects as they move between the fields of view of interconnected cameras.
* Creation of a physical system for application testing: This requirement involves constructing a tangible setup where the application can be thoroughly tested under real-world conditions. This system will include the Nvidia Jetson Nano A02, multiple cameras, and all necessary connectivity hardware to ensure seamless integration and communication between components
* Development of a 3D reality simulation that models object movement within a warehouse: This requirement specifies the creation of a virtual 3D environment that accurately simulates the movement of objects within a warehouse setting

To accomplish the goal of more adaptive and responsive management, a video AI app that is based on NVIDIA DeepStream for real-time analytics and object tracking is developed. This solution emphasizes real-time analytics, detection, classification, and tracking of the objects as well as of the individuals on multiple video feeds.

NVIDIA provides the hardware components for this project with its platforms, wherein benefits like being closely coupled with memory ensure many other advantages but also ascend the challenges that most modern HPC system face including scalability and concurrency. All cloud impacts needs to be data modeled so as to avoid the response time overhead and scalable computing for multiple cameras. Thus, we will ensure that the video feed is not experiencing lagging and that the processing of user actions is not experiencing bottlenecks during the regular influx of multiple video feeds.

Top of Form

## Arhitectura aplicației

During the development of the application, I utilized the NVIDIA Jetson Nano hardware board, equipped with JetPack SDK version 5.0. The application is developed on the Linux operating system that comes pre-installed with JetPack. After the NVIDIA Jetson Nano was flashed with JetPack SDK 5.0, I installed the necessary frameworks required for the application's development, including:

* Python 3.0 serving as the primary programming language for developing the application
* Deepsteeam 5.0
* CUDA
* TensorRT

### The General Use Case of the Application

The following diagram provides a general illustration, using icons and images, of the list of actions or steps that define the interactions between the environment and the system.

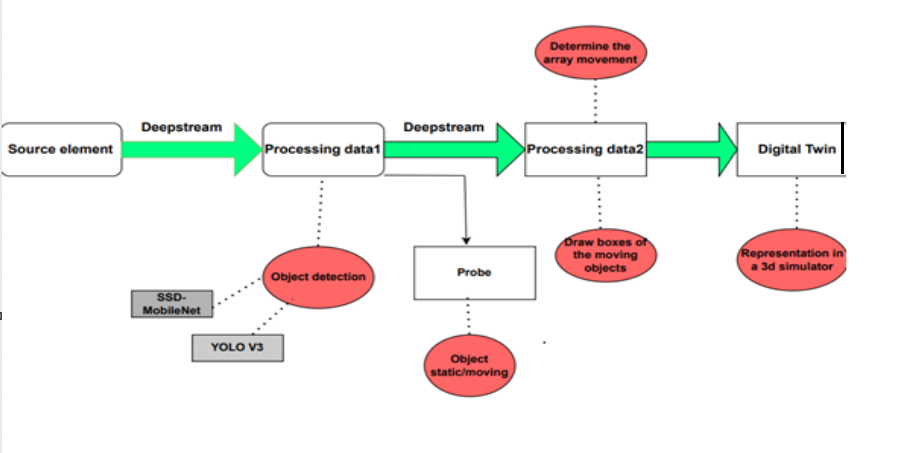


Fig7 Worflow of the application

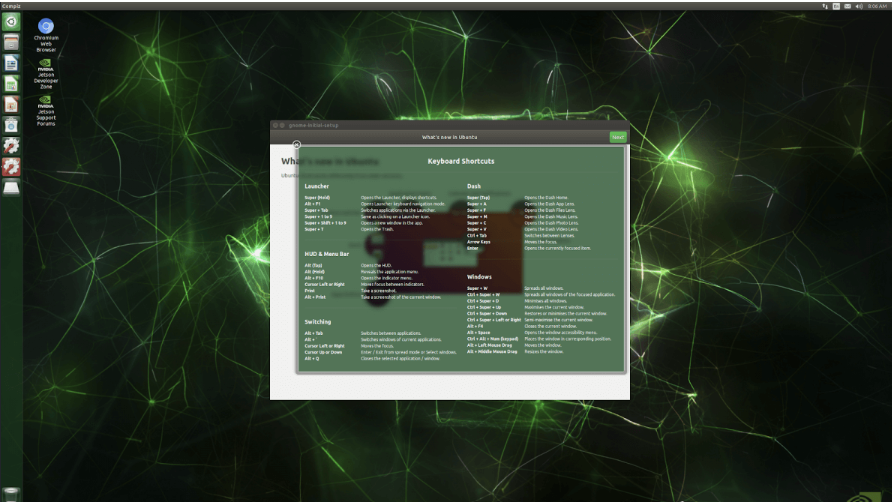
The flow of the system starts from the source elements, which are the cameras video that sends data for the analyzing. The data received goes through two stages of processing. The process of data is done with the using of Deepstream. In the first part of the application, the object detection is done using algorithms like SSD-MobileNet or YOLO (You Only Look Once), these are well-known machine learning algorithms for detection in real time. The model used for the recognition of the object it is MS COCO imagine data set which is able to detect 91 objects.

This extensive training enables it to recognize a wide variety of objects, making it highly versatile for different detection scenarios. The data is used, to inspect data stream, which determines if the object is static or moving. This information is further processed in the second processing stage, which includes to draw boxes on the moving objects and create the array of the movement for the objects in moving. The last stage of the application is to create and update a Digital Twin, which is the virtual representation of the physical system

## Set up the environment

### Set up Jetson Nano board

I set up the NVIDIA Jetson Nano by following several straightforward steps to get the device running smoothly. First, I prepared an SD card (64GB) with the lastest version of the NVIDIA JetPack SDK (version 5.0), which included the operating system (Ubuntu 18.05) and necessary drivers. I downloaded the JetPack image from NVIDIA's official website and used a software tool, Etcher to flash it onto my SD card. Once the SD card was ready and the Jetpack 5.0 was flased to the SD Card, I inserted it into my Jetson Nano. I connected the Nano to a display, keyboard, and mouse so I can use it in display mode. and then powered it up using a 5V 4A DC power supply. In order to power on the Jetson Nano using a barrel jack it is needed to place a power jumper on the pin48. After these steps, the board Jetson Nano was ready for development and deployment of the application.



fig

After JetPack was installed on my Jetson Nano, I installed DeepStream 5.0. This is the lastest version of Deepstream that is compatible withh the board Nvidia Jetson Nano that I am using. I downloaded the DeepStream SDK from NVIDIA’s official website. Following the provided instructions, I successfully installed the DeepStream SDK on my Jetson Nano, ensuring that all necessary dependencies were also installed during this process to enable full functionality.

$ sudo tar -xvf deepstream\_sdk\_v5.0.1\_jetson.tbz2 -C /

$ cd /opt/nvidia/deepstream/deepstream-5.0

$ sudo ./install.sh

$ sudo ldconfig

### Configure Development Environment

To set up my development environment on the Jetson Nano, I first installed a range of development tools and libraries that are essential for building applications. This included Python (version 3.0), which is a versatile scripting language that I use for writing quick scripts and developing complex applications, and GStreamer, which is crucial for handling media processing. I also made sure to install any additional bindings and libraries required for Python and GStreamer to ensure seamless integration with other components of my project.

Next, I proceeded to set up the camera systems. I connected one or more cameras to my Jetson Nano, paying close attention to ensure that they were fully compatible with the platform. It was important to check that the cameras were properly recognized by the system to avoid any issues in capturing video streams. For the development of this application, Hikvision cameras were used, which were connected to the Jetson Nano via WiFi connection. This wireless connectivity enabled a flexible and streamlined setup, allowing for easier positioning and integration of the cameras without the constraints of physical cabling. This setup proved essential for enabling the real-time video processing capabilities required for the application's objective.This setup was critical for the development and testing of applications involving video input and real-time processing.

## Development of the application

### Pipeline Components

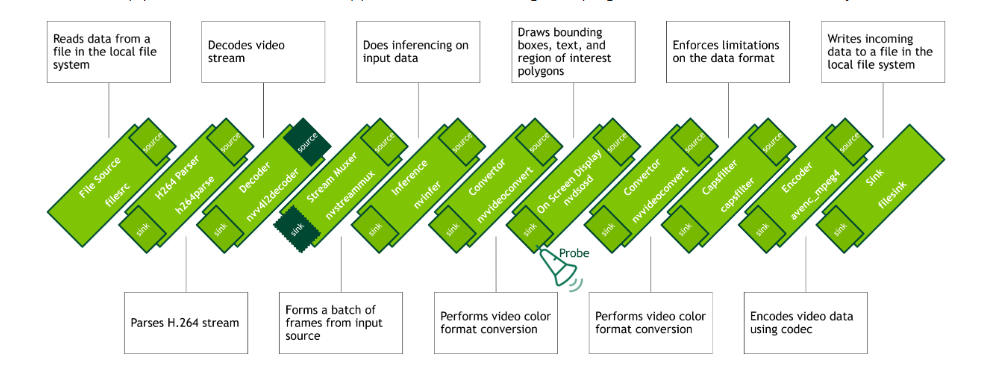


Fig8 Pipeline of the application

* **filesrc** plugin reads data from various types of input sources such as cameras, RTSP streams.
* **h264parse** plugin decodes the basic H.264 stream into frames of data.
* **nvv4l2decoder** plugin decodes this data using the H.264 codec.
* **nvstreammux** plugin aggregates streams, allowing for the handling of multiple input streams and converting them into sequential batched frames. My application processes multiple streams, this plugin is necessary as it supports batched buffers with the NvDsBatchMeta structure.This plugin uses width, height, and batch-size as parameters.
* **nvinfer** plugin processes the input frame (format conversion and scaling) based on network requirements and delivers the processed data to a low-level library. This plugin helps define the deep learning tasks for application by pre-processing frames (converting them into RGB/BGR/GRAY formats, and sending them to the TensorRT engine for inference. This plugin configures my COCO detection model and appends the results (like object classes, bounding box coordinates.) to the metadata.
* **nvvideoconvert** plugin adjusts frames from NV12 (YUV) to RGBA as required by nvdosd and can perform transformations like scaling, cropping, and rotation.
* **nvdosd** plugin overlays bounding boxes and text annotations on the video, drawing from metadata. It requires an RGBA buffer and can convert frames from RGBA to I420 (YUV) for the avenc\_mpeg4 plugin.
* **capsfilter** plugin, while not altering data, enforces specific data format constraints, ensuring the transition from RGBA to I420 (YUV).
* **avenc\_mpeg4** plugin encodes the I420 formatted frames using the MPEG4 codec.
* **filesink** plugin manages incoming data storage to the local file system.

### Initializing Pipeline

As mentioned, it is used the Gvstream plugin. I initialized GStreamer with **Gst.init(list=None)** and instantiate a **Gst.Pipeline** object as pipeline to contain all the elements shown in the pipeline diagram.

import gi

gi.require\_version('Gst', '1.0')

from gi.repository import GObject, Gst, GLib

import pyds

# Initialize GStreamer

Gst.init(None)

pipeline=Gst.Pipeline()

print('Created pipeline')

### Creating pipeline components

Each element in the pipeline was created using Gst.ElementFactory.make(factoryname, name) Elements were configured by setting properties via set\_property(property\_name, value) (refer to the documentation), according to the necessary parameters. In this phase, the elements were also added to the pipeline using Gst.Pipeline.add(element)

|  |
| --- |
| fakesink=pipeline.get\_by\_name('fakesink')  pipeline.remove(fakesink)  nvvidconv1=Gst.ElementFactory.make("nvvideoconvert", "convertor1")  nvosd=Gst.ElementFactory.make("nvdsosd", "onscreendisplay")  nvvidconv2=Gst.ElementFactory.make("nvvideoconvert", "convertor2")  capsfilter=Gst.ElementFactory.make("capsfilter", "capsfilter")  caps=Gst.Caps.from\_string("video/x-raw, format=I420")  capsfilter.set\_property("caps", caps)  encoder = Gst.ElementFactory.make("avenc\_mpeg4", "encoder")  encoder.set\_property("bitrate", 2000000)  filesink=Gst.ElementFactory.make('filesink', 'filesink')  filesink.set\_property('location', 'output\_03\_encoded.mpeg4')  filesink.set\_property("sync", 1)  print('Created elements') |

Even though the source videos were 882 x 692, the nvstreammux plugin required the output dimensions to be set to the nearest multiple of 8, which is why it was necessary to adjust them to 888 by 696. This adjustment would typically be made automatically. Any additional pixels were padded black. The batch-size was set to 1 frame per batch. It is recommended that the batch-size be set equal to the number of input sources.

### Linking and Adding to pipeline

After that all the plugins were added to the pipeline and linked for a optimal worflow

pipeline.add(nvvidconv1)

pipeline.add(nvosd)

pipeline.add(nvvidconv2)

pipeline.add(capsfilter)

pipeline.add(encoder)

pipeline.add(filesink)

print('Added elements to pipeline')

pgie=pipeline.get\_by\_name('primary-inference')

# Link elements together

pgie.link(nvvidconv1)

nvvidconv1.link(nvosd)

nvosd.link(nvvidconv2)

nvvidconv2.link(capsfilter)

capsfilter.link(encoder)

encoder.link(filesink)

print('Linked elements in pipeline')

Finally, all the elements were linked in the sequence that allows data to flow through the pipeline,usingGst.Element.link(Gst.Element). Initially, the nvinfer plugin, which is crucial for deep learning inference, was retrieved by its name 'primary-inference' from the pipeline identified as pgie. The output from the inference plugin (pgie) was routed to a video conversion element (nvvidconv1), which prepared the video for subsequent object detection (nvdosd).

Following detection, the video stream underwent further conversion to align with processing or output requirements by another converter (nvvidconv2). The video stream was then processed through a caps filter (capsfilter), where specific parameters were set before the stream was encoded by the encoder (encoder). Ultimately, the output from the encoder was connected to a file sink (filesink), responsible for saving the processed video to the filesystem. A confirmation message was printed to ensure that all elements within the pipeline were correctly linked.

### Starting the pipeline

The pipeline was equipped with a bus that I used to monitor messages. I initiated a GLib/Gtk+ MainLoop (or consistently iterated the default GLib main context) and attached a watch/message handler to the bus with **Gst.Bus.add\_signal\_watch()**. This configuration ensured that the GLib.Mainloop regularly checked the bus for new messages and notified me accordingly. Message handling was facilitated through a callback function I defined as bus\_call. This function was invoked each time the pipeline sent a message to the bus and needed to return True to remain attached.

loop=GLib.MainLoop()

bus=pipeline.get\_bus()

bus.add\_signal\_watch()

bus.connect("message", bus\_call, loop)

print('Added bus message handler')

print("Starting pipeline")

pipeline.set\_state(Gst.State.PLAYING)

try:

    loop.run()

except:

    pass

pipeline.set\_state(Gst.State.NULL)

Due to its frequent use across various DeepStream testing that were done, this callback function for monitoring bus messages was refactored into a Python script (bus\_call.py) to simplify usage. When I used inspect.getsource(object) to examine the function's definition, it showed a straightforward procedure that evaluated the message type and processed them appropriately.

With the message handler in place, I set the pipeline to the PLAYING state and ran the MainLoop. Finally, when the pipeline's tasks were completed, I transitioned it into the NULL state to ensure proper cleanup.

|  |
| --- |
| import gi  import sys  gi.require\_version('Gst', '1.0')  from gi.repository import GObject, Gst  def bus\_call(bus, message, loop):      t = message.type      if t == Gst.MessageType.EOS:          sys.stdout.write("End-of-stream")          loop.quit()      elif t==Gst.MessageType.WARNING:          err, debug = message.parse\_warning()          sys.stderr.write("Warning: %s: %s\n" % (err, debug))      elif t == Gst.MessageType.ERROR:          err, debug = message.parse\_error()          sys.stderr.write("Error: %s: %s\n" % (err, debug))          loop.quit()      return True |

### Detection of objects