FEDERAL UNIVERSITY OF ITAJUBÁ

IESTI05 - MACHINE LEARNING SYSTEMS ENGINEERING

Part 2 Final Project: GenAI At the Edge

Theme: Smart Inspection

Author: Eduardo José de Souza Castro

ITAJUBÁ

2025

1. INTRODUCTION:

This project aims to validate the knowledge acquired in Part 2 of the Edge AI Engineering course offered by the Institute of Systems Engineering and Information Technology (IESTI) at the Federal University of Itajubá (UNIFEI).

The project introduces a scenario of inspection and monitoring of an indoor environment using interactive tools and real-time responses from physical actuators. The motivation arises from the high demand for intelligent and autonomous monitoring across different contexts. The distinguishing aspect of this approach lies in its ability to interact with the user in multiple environments and in the use of devices with limited yet sufficient resources, which directly influences the overall project cost.

In this context, it was possible to apply resources related to generative AI, such as SLMs, deployed on edge devices capable of processing prompts using open-source models executed through a powerful command-line tool called '*ollama*'. With this setup, the software framework was integrated with physical sensors and actuators, including motion sensors, lidars, drones and indicator LEDs.

The general idea is to use a motion sensor to monitor a room that simulates an application environment and respond to user commands. If any activity is detected, the possibility of performing a drone inspection is presented to the user for evaluation. This inspection returns data from a deck mounted on the drone containing five distance sensors, which provide a more accurate metric for determining whether any anomaly is present in the environment.

The elements involved in the process were selected from the equipment available in the university laboratories and could be rearranged as needed to better suit the specific application, provided that compatibility requirements and the standardisation of the embedded system supporting the functionalities were respected.

Thus, the criteria adopted for evaluating the project results relate, first, to the effective analysis by the SLM of the data reported by the sensors, the appropriate responses to the previously established directives, the coherent interaction with the user and, finally, the correct and effective manipulation of the actuators based on the interpretation generated by the model in execution.

2. CONTEXT AND LITERATURE:

Research in edge AI and IoT has expanded considerably in recent years, driven by the need for autonomous systems capable of operating with low latency and under strict resource constraints. Several established projects illustrate the evolution of this field. The TinyML initiative, for example, consolidates methods for deploying machine learning models on microcontrollers with very limited memory, demonstrating the feasibility of local inference in applications such as anomaly detection, environmental monitoring and gesture recognition. Platforms like Edge Impulse further extend this approach by offering a complete workflow for data collection, training and deployment of optimised models directly on embedded hardware. Similar efforts appear in the Google Coral ecosystem, which integrates specialised TPUs for real-time inference at the edge, and in NVIDIA Jetson-based projects focused on robotics and autonomous navigation in constrained environments.

Within this broader landscape, the adoption of small language models (SLMs) on edge devices has begun to gain prominence. Although more recent than vision-based edge workloads, SLM deployments have been explored in systems that require local natural-language interpretation without reliance on cloud services. Projects using open-source models such as LLaMA-based derivatives or Phi-2 running through lightweight runtimes like '*ollama*' or '*llama.cpp*' highlight the practical limits and capabilities of these architectures. The literature shows consistent results in tasks such as command parsing, classification and contextual reasoning, provided that prompt structures and model sizes remain aligned with the hardware's available compute and memory. These models, however, present clear constraints in long-horizon reasoning and in scenarios that demand high robustness to ambiguous inputs, which reinforces the need for careful model selection in embedded contexts.

Regarding optimisation techniques, studies related to TinyML, ONNX Runtime for Mobile and TensorRT for Jetson platforms consistently emphasise quantisation, pruning and parameter reduction to enable efficient deployment on edge hardware. These approaches decrease computational load and memory usage while preserving acceptable accuracy. Research from the MLPerf Tiny benchmark also demonstrates how model compression strategies influence inference time and

energy consumption in embedded systems. The combination of these methods has become essential for edge AI solutions that couple perception, interpretation and actuation in real time, particularly when integrating heterogeneous sensors, embedded controllers and physical actuators, as required in this project.

Considering this range of tools, there has been a consistent increase in the adoption of generative AI models in applications that involve direct interaction with the physical environment. This trend requires architectures capable not only of interpreting sensor data but also of performing appropriate actions based on user interactions expressed in natural language. In this way, the model becomes central to integrating perception, interpretation and action within embedded systems.

3. SYSTEM DESIGN:

3.1 Hardware:

- **Single-board Computer:**

For the use of the generative AI resources required for this project, it was necessary to select a device capable of running an operating system that supports the toolchain while providing specifications that were sufficient for the intended workload. For this purpose, a Raspberry Pi 5 equipped with a BCM2712 SoC featuring four Cortex-A76 cores at 2.4 GHz and 8 GB of RAM was employed.

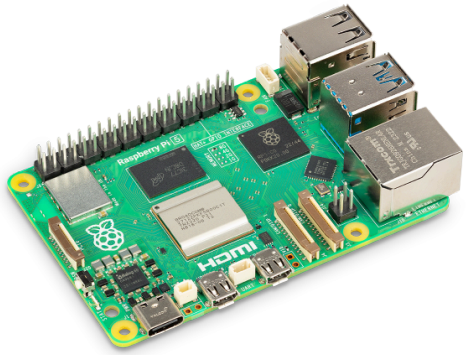

Figure 01: Raspberry PI 5

The Raspberry Pi 5 served as the central processing unit of the system, providing both the computational capability required for running the generative AI models and the necessary interfaces for sensor and actuator integration. The board's general-purpose input and output (GPIO) pins enabled direct communication with the motion sensor, allowing the system to capture real-time physical events and forward them to the SLM for interpretation. Through these pins, the Raspberry Pi monitored digital state changes, which triggered subsequent decisions and actuation commands.

The device's USB ports were also employed for interaction with the drone, enabling the exchange of command signals and the reception of data from sensors attached to the platform.

- **Motion Sensor:**

The sensor used here was The HC-SR501 is a PIR-based motion sensor designed for automatic control applications, built around the LHI778 infrared probe, offering high sensitivity, high reliability and low-power operation for battery-driven systems. It operates from 5 to 20 V, features a low static current draw, provides a 3.3 V TTL output and supports both repeatable and non-repeatable trigger modes, with an adjustable sensing range of up to 7 meters within approximately 120 degrees.

Figure 02: PIR HC-SR501 motion sensor

The module includes adjustable delay and distance settings, optional photosensitive and temperature-compensation features and an internal blocking time to prevent false triggers. After a brief initialization period on power-up, it enters a stable standby state and performs best when installed such that human movement crosses the sensor laterally, where differential infrared detection is maximized. Its compact dimensions and straightforward electrical interface make it suitable for automated lighting, alarms, ventilation systems and general-purpose motion detection in embedded environments.

- **DRONE:**

The drone used in this task was the Crazyflie Drone, more specifically the Crazyflie 2.1+, developed by Bitcraze Company. This drone is a lightweight open-source flying development platform designed for research and experimentation in constrained environments. Weighing only 29 g, it integrates both low-latency long-range radio and Bluetooth LE, enabling flexible communication either through mobile devices or through a computer using Crazyradio interfaces for more advanced control and data access.

Figure 03: Crazyflie 2.1+ drone

The 2.1+ version includes an upgraded battery and improved propellers, enhancing flight performance by up to 15 percent. Supported by a broad software ecosystem and modular expansion decks, it provides a stable and extensible platform suitable for tasks such as indoor inspection, sensor integration and autonomous behaviour development within embedded AI workflows.

To interface the drone with the system, the Crazyradio 2.0 was used as the primary communication bridge. This equipment is a long-range open USB radio dongle built around the Nordic Semiconductor nRF52840, equipped with a 20 dBm power amplifier and an LNA to enhance communication reliability in indoor environments. Although commonly used within the Crazyflie ecosystem, its open firmware, Python API and stable low-latency radio link make it a suitable component for broader embedded applications that demand predictable timing without the bandwidth requirements of Wi-Fi.

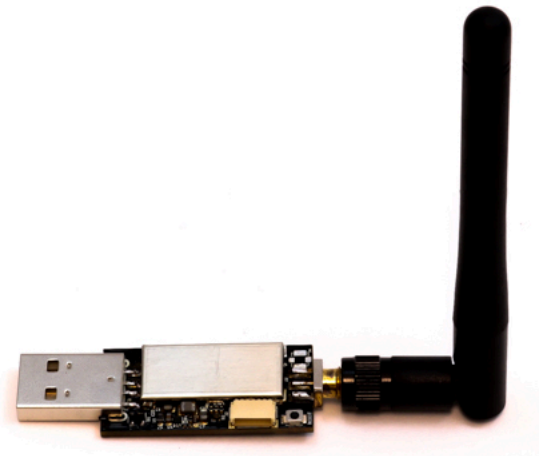

Figure 04: Crazyradio 2.0

In this project, the Crazyradio 2.0 was connected to the Raspberry Pi 5 through one of its USB ports, enabling a direct and stable communication channel between the drone and the system for command transmission and sensor data reception.

In addition, two decks are mounted on the drone, each serving a distinct purpose within the sensing and navigation pipeline. The Flow Deck V2 provides the core positioning capability required for stable flight, while the Multi-ranger Deck offers directional distance sensing through its five lidar units.

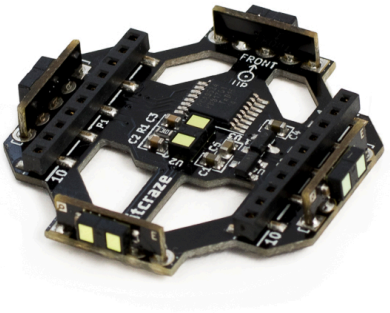

Figure 05: Multiranger Deck

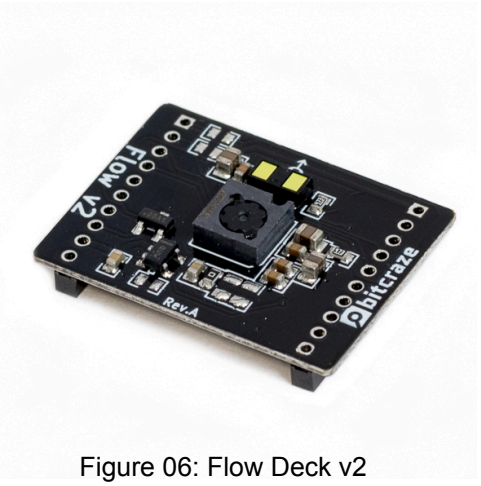

Figure 06: Flow Deck v2

The Flow Deck V2 combines a VL53L1X time-of-flight sensor for ground-distance estimation with an optical flow sensor that measures horizontal motion relative to the surface. This configuration enables the Crazyflie 2.x to maintain stable hovering, execute controlled maneuvers and support pre-defined trajectories. The Multi-ranger Deck complements this functionality by extending the drone's spatial awareness; its five lidars measure the distance to obstacles in the front, back, left, right and upward directions with millimetre-level precision up to 4 meters. Together, these sensing modules allow the platform to perform indoor navigation, obstacle detection and environment-aware tasks with greater reliability.

- **Indicator LEDs:**

Two indicator LEDs, one green and one red, were integrated into the system to provide a clear and immediate response to the user following the drone inspection.

Figure 07: Green and red LEDs

After the sensor data collected by the drone is processed and interpreted by the SLM, the system activates the appropriate LED to signal the outcome: the green LED indicates that no anomaly was detected in the inspected environment, while the red LED signals the presence of a potential issue. This visual feedback mechanism ensures that the user receives a straightforward and unambiguous assessment of the inspection results.

3.2 Software:

The embedded software was developed in an organised and modular manner, with its functionalities clearly separated and structured according to the object-oriented programming paradigm. Python, the high-level language commonly used in embedded prototyping and rapid development, served as the foundation for implementing these components.

The system operates through well-defined modules that interact to form the final application, which imports all required classes and subsystems. This organisation relied on software-engineering principles such as abstract classes, inheritance, encapsulation through public and private methods and properly structured getters and setters. It also required the use of Python-specific features, including module creation, path resolution for imports and the separation of responsibilities across files and packages.

The data flow was designed so that each subsystem is represented by a dedicated class responsible for its configuration, lifecycle management and method structure. Sensor-related operations are encapsulated in classes such as *"PIRMotionDetector"*, *"DHT22"*, *"BMP280"* and *"ButtonSensor"*, all derived from the abstract base class *"Sensor"*. Actuation tasks are centralised in the *"BasicActuators"* class, while drone communication and motion commands are handled by *"CrazyflieActuator"*. The interaction pipeline with the small language model is governed by *"SLMConfig"* and *"InteractivityHandler"*, which prepare prompts, interpret model responses and enforce JSON-based control rules. At the core of the system, the *"SmartInspection"* class orchestrates the entire workflow: it aggregates sensor readings, forwards system status to the SLM, interprets the returned intent, updates LED states and triggers drone inspection routines when required. This structured exchange of information between classes ensures a coherent execution pipeline in which each module contributes to the final autonomous inspection behaviour.

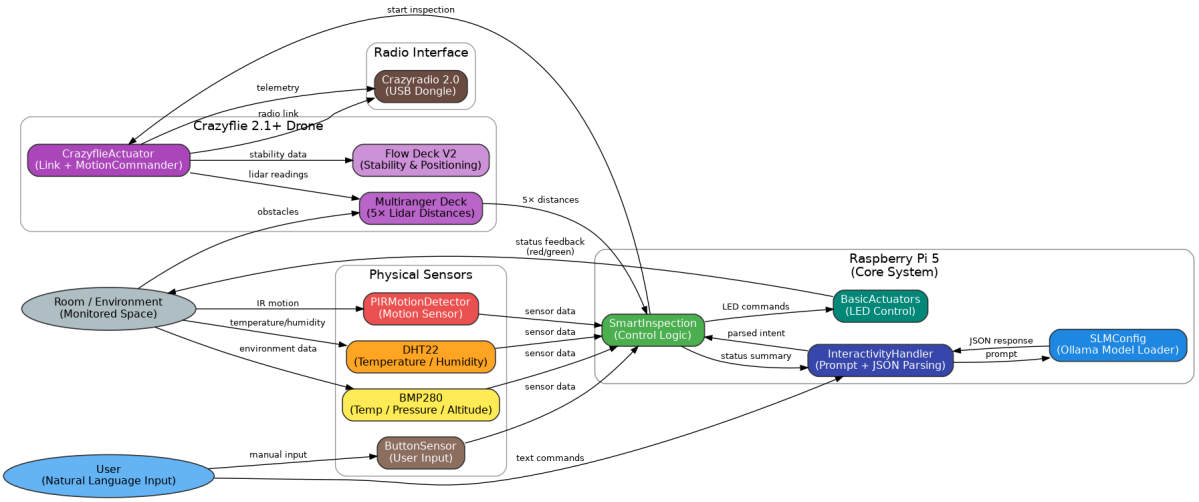

Figure 08: dataflow architecture

Figure 08 presents the high-level data-flow architecture of the project, highlighting how information moves between the different software modules. Sensor readings are captured by classes such as "PIRMotionDetector", "DHT22" and "BMP280" and forwarded to the central controller "SmartInspection". This class communicates with the small language model through "InteractivityHandler" and "SLMConfig", which process user queries and system prompts. Decisions returned by the SLM then propagate to "BasicActuators" for LED control or to "CrazyflieActuator" to initiate a drone inspection. The diagram visually reinforces the modular design of the system and the orderly flow of information that enables autonomous interaction and decision-making.

Software Dependencies and Libraries

The embedded inspection system depends on a set of external packages, each responsible for a specific function within the application. The *"Adafruit-Blinka"* stack and its associated CircuitPython modules handle communication with environmental sensors. The GPIO interfaces rely on *"gpiozero"* and *"lgpio"* for controlling input and output devices. Drone communication and motion control are managed through *"cflib"*. Finally, small-language-model inference is executed locally using *"ollama"*, which enables the system to process user instructions and generate structured responses.

- Dependencies and version:

  Adafruit-Blinka==8.67.0
  adafruit-circuitpython-bmp280==3.3.9
  adafruit-circuitpython-dht==4.0.10
  adafruit-circuitpython-busdevice==5.2.14
  adafruit-circuitpython-register==1.11.1
  gpiozero==2.0.1
  lgpio==0.2.2.0
  cflib==0.1.29 (numpy==2.3.4)
  ollama==0.6.0

The dependencies include more libraries than those strictly required for the core scope of the project. This is because the system is designed to accommodate additional sensors and components, allowing it to support a wide range of potential applications. In addition, library versions such as *"numpy"* may introduce compatibility conflicts. Therefore, it is recommended to use a dedicated Python virtual environment for this project, ensuring that all libraries and their corresponding versions remain isolated and consistent, preventing potential dependency issues.

Entering the context of SLMs within the project, the *"llama3.2:3b"* model was chosen for its favourable balance between size and capability, making it suitable for execution on edge hardware while still providing reliable natural-language interpretation. With a 3-billion-parameter architecture, it offers enough expressiveness to perform command parsing, short-context reasoning and structured response generation without exceeding the system's computational limits. Its consistent ability to produce well-formed JSON outputs ensures deterministic behaviour, which is essential for mapping language instructions to actuator control and inspection routines. Moreover, the model achieves fast inference times when deployed in quantised form, enabling responsive interaction during real-time operation. These characteristics collectively justify its selection as the SLM driving the inspection workflow.

All system software, along with documentation and docstrings, can be reviewed in the GitHub repository: Smart Inspection Repository.

4. IMPLEMENTATION

The implementation followed a sequential workflow, beginning with the configuration of the Raspberry Pi 5 using the official 64-bit Raspberry Pi OS with a graphical interface. In parallel, the required sensors and the associated software tools were installed on the system to ensure full operational capability. The motion sensor and the Crazyradio module were successfully integrated into the setup, along with a few additional test sensors that were used during early validation but were not part of the final project.

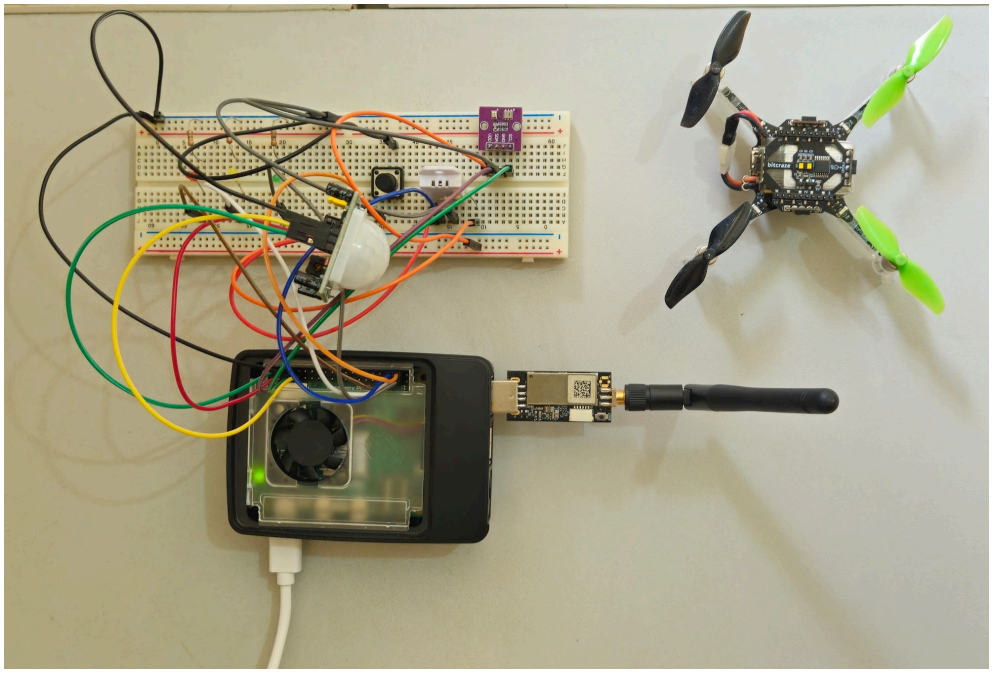

Figure 09: complete raspberry setup

Below are the pinout tables for the constructed circuit:

Motion Sensor HC-SR501	
Sensor PIN	**Raspberry PIN**
5V	PIN 2 (5V)
GND	PIN 6 (Ground)
TTL output	PIN 7 (GPCLK0)

LEDs Actuators	
LEDs	**Raspberry PIN**
Red	PIN 33 (GPIO13)
Green	PIN 37 (GPIO26)
Yellow	PIN 35 (GPIO19)

For the Crazyradio, any USB port can be used to maintain communication with the drone, provided that the USB permission settings in Linux are updated accordingly, as can be verified in the referenced link: USB Permissions Crazyradio.

In addition, to attach the decks to the Crazyflie, the Bitcraze documentation is highly comprehensive and was essential for configuring the drone. Information about these decks, as well as others, can be found at the following link: Expansion Decks.

With respect to the software structure of the project, all components were designed to be scalable, reusable and robust against both user errors and internal system faults. An example of this approach is the abstract class *"Sensor"*, which defines essential and standardised methods that guide the implementation of any sensing device within the project. This ensures uniform behaviour across sensor modules and facilitates the addition of new hardware without disrupting the existing workflow.

```python
class Sensor(ABC):

    """
    Abstract class to define the sensors behavior and methods
    """

    def __init__(self, name: str) -> None:

        """

        :param name(str): name of the sensor. Default is the class name.

        """

        self.configured = False
        self.name = name

    @abstractmethod
    def initial_config(self) -> None:

        """
        Setup specifics sensor settings. It calls protected function _setup
        that must be implemented in inherited classes.

        """
        if not self.configured:
            try:
                self._setup()
            except Exception as ex:
                print(f"Erro has occured while setting up sensor: {ex}")
                return
            else: self.configured = True

        else: print("Device already configured")

    @abstractmethod
    def _setup(self) -> None:
        raise NotImplementedError(f"Should be implemented setup method for class")
```

Figure 10: Part of Sensor abstract class implementation

The same design philosophy extends to the actuator and drone interfaces, where classes such as *"BasicActuators"* and *"CrazyflieActuator"* encapsulate their respective operations behind well-defined APIs. Additionally, the interaction layer implemented through *"InteractivityHandler"* and *"SLMConfig"* provides controlled input processing and error handling, preventing malformed commands or unexpected model outputs from propagating through the system.

The SLM was integrated through a dedicated class, *"SLMConfig"*, which handles model initialisation and establishes a predefined system prompt that dictates the model's behaviour, expected output format and operational constraints. This ensures that the model begins every interaction already aware of its role and the functions available within the system.

The integration logic also manages the message flow between the user, the model and the higher-level controller, maintaining consistent communication throughout the application. The optimisation strategy relied on selecting the compact *"llama3.2:3b"* model, which operates efficiently on the Raspberry Pi 5 while retaining sufficient capability for command interpretation. Additional optimisation was incorporated into the system prompt, which restricts the model's behaviour, standardises output through JSON formatting and reduces unnecessary reasoning steps.

The function-calling mechanism implemented in *"SLMConfig"* improves reliability by ensuring that action-related outputs are only produced when explicitly required. This mechanism is controlled through the *tools* variable defined within *"SLMConfig"*, which specifies whether the function responsible for initiating the drone inspection is available to the model. In the *"SmartInspection"* main application, this variable is initially set as an empty list, meaning no function calls can be triggered by default. The system then selectively enables the drone-inspection function by adding it to the *tools* list only when the user prompt contains specific keywords such as "drone", "fly" or "crazyflie". This conditional activation prevents the model from invoking the function-calling mechanism unnecessarily and ensures that the drone inspection routine is executed solely when explicitly requested by the user. The prompt engineering strategy remains centred on a structured system prompt defining strict output rules and control fields, resulting in a deterministic interaction model aligned with the operational requirements of the inspection workflow.

5. RESULTS AND DISCUSSION

All sensing components were successfully validated, with consistent readings obtained from the environmental sensors, the motion detector and the Multi-ranger Deck. The system correctly captured motion events and returned reliable distance measurements in all tested directions, confirming the functional integration of both the Raspberry Pi sensors and the drone-mounted deck.

The response time of the SLM model remained within acceptable bounds for the intended interaction loop, with complete processing—including sensor interpretation and model output—remaining below approximately 90 seconds. Although not instantaneous, this latency did not compromise the system's ability to maintain coherent interaction with the user. The overall interpretation quality of the SLM remained stable once the prompt structure and output format were adjusted through JSON-based constraints.

The inspection routine was also validated. Upon motion detection by the presence sensor, the system prompted the user for confirmation before initiating the drone inspection sequence. Alternatively, the drone could be triggered directly through textual commands containing explicit keywords such as "drone", "crazyflie" or "fly", as handled by the interactive interface. During testing, however, an issue was identified in which the model triggered the drone inspection in response to any prompt interpreted as an action, regardless of its relevance. This behaviour was corrected through the use of function calling with guarded activation: the inspection function is now invoked only when the user input contains one of the predefined trigger words. This adjustment eliminated unintended drone activations and stabilised the interaction logic.

In relation to the drone behaviour itself, the Crazyflie platform demonstrated adequate responsiveness to the inspection routine, though its inherent flight stability limitations occasionally introduced minor inconsistencies. Despite this, the demonstration flights executed the take-off, rotation and landing phases as expected, validating the feasibility of the inspection component within the constraints of the selected hardware.

Overall, the primary issues encountered were associated with uncontrolled function calling and the model's initial tendency to treat arbitrary action-like prompts

as requests for drone flight. After modifying the activation logic, the model consistently adhered to the expected behaviour. Sensor readings remained reliable throughout testing, and the system operated as intended, with coherent model interpretation, predictable LED responses and a functional drone inspection routine. The resulting implementation delivered an effective interaction loop with reasonable latency and correct execution of all system modules.

6. FUTURE WORKS AND CONCLUSION

The project demonstrated that the core inspection workflow operates reliably, with all sensors returning validated readings and the SLM producing consistent interpretations once the prompt structure and function-calling conditions were refined. Issues encountered, such as the model triggering drone actions in response to unrelated prompts, were resolved through controlled activation of the tools list and explicit keyword checks, resulting in a stable and predictable interaction loop. Although the Crazyflie platform introduced some instability during flight, the inspection routine remained functional and sufficient for validating the concept. In terms of future work, a natural extension involves exploring the use of RAG-based techniques to generate structured datasets from full room inspections. Initial tests were conducted to produce a CSV dataset using the five distance readings from the Multi-ranger Deck, forming a table that could be used to retrieve previously recorded spatial information.

Front	Back	Right	Left	Up	Status
2.65	0.68	0.77	1.85	1.54	Nothing detected
2.65	0.69	0.75	1.86	1.51	Nothing detected
2.61	0.7	0.82	1.87	1.5	Nothing detected
2.02	0.74	2.6	0.75	1.49	Nothing detected
1.97	0.76	2.56	0.75	1.48	Nothing detected
1	1.08	1.71	0.79	1.48	Nothing detected
0.8	2.52	2.1	0.65	1.49	Nothing detected
0.6	1.86	0.86	0.99	1.51	Nothing detected
0.41	2.27	0.73	1.28	1.51	Nothing detected
0.42	2.25	0.71	2.62	1.51	Nothing detected
1.05	0.76	0.35	1.48	1.53	Nothing detected
2.59	0.74	0.41	2.29	1.54	Nothing detected
2.57	0.77	0.43	2.31	1.54	Nothing detected

Figure 11: CSV dataset with Multiranger Deck data

However, early experiments indicated that RAG is not well suited for interpreting numerical variations, as the model showed limitations in understanding changes in distance values and deriving meaning from them. Even so, the creation of such datasets remains valuable, as they could enable comparison between new inspections and historical patterns to help identify potential anomalies. Another possible direction includes aggregating sensor data over time and training a dedicated model, using frameworks such as TensorFlow, to perform anomaly detection or independent inference based on these numerical readings. This would create a complementary evaluation layer that operates alongside the SLM. Collectively, these approaches may enhance system robustness by combining conventional machine-learning methods with language-driven reasoning.

From a scalability perspective, the current architecture is well prepared for expansion, as the modular design and the use of abstract classes allow new sensors, actuators or behaviours to be integrated with minimal structural changes. Overall, the system proved functional within its defined scope, and the outlined improvements offer clear pathways for expanding both capability and reliability in future iterations.

7. ACKNOWLEDGEMENT

The authors express their gratitude to the VISCAP Group (Grupo de Visão, Sistemas de Computação e Aplicações) of the Institute of Mathematics and Computing (IMC) at UNIFEI, coordinated by Professor Alexandre Carlos Brandão Ramos, for providing the Crazyflie platform, the associated decks and the Crazyradio module, which were essential for making this project possible.

8. REFERENCES

ROVAI, Marcelo. IESTI05 – Edge AI: Machine Learning System Engineering Course. UNIFEI – Instituto de Engenharia de Sistemas e Tecnologia da Informação. Material available at: https://github.com/Mjrovai/UNIFEI-IESTI05-EDGE_AI.git

HOW TO INTERFACE HC-SR501 PIR SENSOR WITH RASPBERRY PI. Hackaday. Available at: https://hackaday.io/page/8804-how-to-interface-hc-sr501-pir-sensor-with-raspberry-pi . Accessed on: Nov, 27. 2025.

TWOBITCIRCUS. How to Use a PIR Motion Sensor with Raspberry Pi. YouTube, 2020. Available at: https://www.youtube.com/watch?v=Tw0mG4YtsZk. Accessed on: Nov, 27. 2025.

BITCRAZE. Documentation Portal. Available at: https://www.bitcraze.io/documentation/start/. Accessed on: Nov, 27. 2025.

PYTHON SOFTWARE FOUNDATION. threading — Thread-based parallelism. Python Documentation. Accessed at: https://docs.python.org/3/library/threading.html. Accessed on: Nov, 27. 2025.

GPIOZERO. *InputDevice — GPIO Zero documentation*. Available at: https://gpiozero.readthedocs.io/en/stable/api_input.html#gpiozero.InputDevice. Accessed on: Nov, 27. 2025.