# Vision-Language Models for Robot Control

#### Author,

Vivek DR

Robotic Sr Software developer Asimov Robotics Pvt Ltd

## Table of Contents

**Objectives for Vision-Language Models for Robot Control Chapter 01: Why Native Installation Over Docker?**

**Chapter 02: Installing Ollama on Jetson Orin**

**Chapter 03: Camera Connection with Jetson Orin Nano Using GStreamer Pipeline** **Chapter 04: Integrating Camera and Ollama with ROS 2 Humble**

## Objectives for Vision-Language Models for Robot Control

#### Final Goal: Intelligent Robot Interaction with Objects Based on User Instructions

* + Achieve a system where the robot can detect objects, understand user commands, and interact with objects accordingly for tasks like pick-and-place and navigation.

#### Deploy a Vision-Language Model on Jetson Orin Nano

* + Install and optimize a pre-trained Vision-Language Model (VLM) for efficient on- device processing.

#### Integrate VLM with ROS 2, Camera Systems, and Navigation Stack

* + Enable seamless communication between the VLM, ROS 2, and camera systems for real-time image processing.
  + Begin with a webcam for initial testing and transition to a depth camera for enhanced precision in pick-and-place and navigation tasks.

#### Enable Object Detection and Image Segmentation

* + Implement image segmentation and refine the VLM model to recognize objects accurately.
  + Ensure the system understands user commands related to detected objects.

#### Generate and Execute ROS 2 Action Sequences from Natural Language Commands

* + Convert user instructions into corresponding ROS 2 action sequences for tasks like pick-and-place and navigation.
  + Ensure the robot executes actions based on natural language inputs.

#### Integrate MoveIt for Robotic Arm Control

* + Set up MoveIt with the robotic arm to perform precise pick-and-place actions guided by VLM-generated commands.

#### Achieve Real-Time, Reliable Operation

* + Optimize system response times for seamless and safe robot control.
  + Ensure robustness in dynamic environments.

#### Enable Pick-and-Place and Navigation via Natural Language

* + Allow users to control pick-and-place actions and navigate using spoken or written commands.
  + Ensure the robot can detect and interact with objects based on user instructions.

## Chapter 01: Why Native Installation Over Docker?

### Introduction

When setting up Ollama on Jetson Orin, there are two main approaches: **Native Installation** and **Docker-based Installation**. While Docker provides convenience and ease of setup, native installation offers better system integration and performance optimization.

### Advantages of Native Installation

1. **Direct Hardware Access**: Running Ollama natively allows direct interaction with Jetson’s GPU, ensuring optimal CUDA and TensorRT performance without additional overhead.
2. **Better Resource Utilization**: Containers introduce some level of abstraction, which can lead to inefficiencies in resource allocation. Native installation ensures that system memory, CPU, and GPU resources are used efficiently.
3. **Seamless Integration with ROS2 and Other Tools**: If you’re running ROS2 and Drogon, a native installation ensures that all dependencies work without container-specific networking or storage issues.
4. **Lower Latency**: Since there is no containerization overhead, native execution ensures faster model inference, which is critical for real-time applications such as robotics.
5. **Persistent Storage and Simplicity**: With Docker, managing persistent data (like downloaded models) requires explicit volume mounts. Native installation avoids this complexity.

### Challenges of Native Installation

* + **Dependency Management**: Unlike Docker, where dependencies come pre-configured, native installation requires manual setup of CUDA, cuDNN, and other necessary libraries.
  + **Potential Conflicts**: Installing different versions of Python, PyTorch, or TensorRT may cause conflicts with other applications.

### Comparison Table: Native vs. Docker

|  |  |  |
| --- | --- | --- |
| **Feature** | **Native Installation** | **Docker Installation** |
| GPU Performance | Full access | ⚠ Slight overhead |
| Ease of Setup | Manual dependency setup | Pre-configured |
| Integration with ROS2 | Seamless | ⚠ Requires additional configuration |
| Latency | Lower | Slight increase |
| Persistent Storage | Simple | ⚠ Requires volume mounts |

**Conclusion**

While Docker provides an easier setup process, native installation is the preferred method for advanced users who want maximum control over hardware performance, system integration, and long-term stability. By following this guide, you will learn how to install Ollama natively and optimize it for real-time robotics applications on Jetson Orin.

## Chapter 02: Native Installation Ollama on Jetson Orin Nano

This chapter provides a comprehensive guide to installing Ollama on Jetson Orin Nano, utilizing the LLaVA:7B model. It also compares this model with other alternatives, highlighting why LLaVA:7B is suitable for Jetson Orin Nano with 8GB RAM.

## Installation Guide

Follow the instructions in the [official Ollama Linux documentation](https://github.com/ollama/ollama/blob/main/docs/linux.md) to install Ollama on Jetson Orin Nano.

### Steps:

#### Update System:

sudo apt update && sudo apt upgrade

#### Install Dependencies:

sudo apt install curl git

#### Download Ollama:

curl -fsSL https://ollama.ai/install.sh | sh

#### Verify Installation:

ollama --version

#### Run LLaVA:7B Model:

ollama run llava:7b

## Why Use LLaVA:7B?

LLaVA:7B is optimized for lightweight hardware like the Jetson Orin Nano with 8GB RAM, providing efficient performance without extensive resources. It offers vision-language capabilities similar to LLaMA 3.2-Vision but with significantly lower hardware requirements.

### Key Advantages of LLaVA:7B:

* + **Hardware Efficiency:** Runs smoothly on 8GB RAM, making it ideal for edge devices like Jetson Orin Nano.
  + **Multimodal Capabilities:** Integrates vision and language tasks effectively, suitable for AI vision applications.
  + **Lower Power Consumption:** Reduces power usage compared to larger models, extending the lifespan of portable setups.
  + **Optimized for Inference:** Provides quick inference times even on resource-limited hardware.
  + **Scalable Performance:** Can handle various tasks, from object detection to natural language processing, without overloading the system.

### Model Comparison Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Feature** | **LLaVA:7B** | **LLaMA 3.2-Vision** | **Mistral 7B** | **Falcon 7B** |
| **RAM Required** | 8GB | 16GB | 12GB | 10GB |
| **Performance** | Fast, lightweight | High, resource-heavy | Balanced | Efficient processing |
| **Hardware Support** | Jetson Orin Nano | High-end GPUs | Mid-range GPUs | Jetson Orin, GPUs |
| **Power Usage** | Low | High | Moderate | Low |
| **Ease of Setup** | Simple manual setup | Complex due to size | Moderate | Simple |
| **Best For** | Edge AI, vision tasks | High-performance AI | General AI tasks | Efficient AI on Orin |

LLaVA:7B is the most suitable choice for Jetson Orin Nano, balancing power and performance. Its ability to run complex AI tasks without requiring expensive hardware makes it highly advantageous.

## Conclusion

Installing Ollama with LLaVA:7B on Jetson Orin Nano allows efficient AI model deployment on edge devices. Its minimal RAM usage, effective performance, and advanced multimodal capabilities make it ideal for lightweight hardware, compared to more resource-intensive models.

# Chapter 03: Camera Connection with Jetson Orin Nano Using GStreamer Pipeline

This chapter provides the setup and configuration for connecting a camera to the Jetson Orin Nano using a GStreamer pipeline. The chosen pipeline addresses issues with high frame rates (60fps) causing hangs due to ARM architecture constraints on the Jetson Orin Nano.

## Connecting the Camera

Ensure the camera is connected to the Jetson Orin Nano via USB or CSI interface.

## Reason for GStreamer Pipeline Choice

The Jetson Orin Nano (8GB) experiences performance issues with high frame rate video (such as 60fps) when using direct OpenCV access (cv2.VideoCapture('/dev/video0')). This is due to ARM architecture limitations and lack of optimized drivers for high FPS streaming.

GStreamer pipelines offer a stable alternative by handling video capture more efficiently through hardware-accelerated plugins.

### Recommended GStreamer Pipelines:

1. **Basic GStreamer Pipeline**
   * **Pipeline**: v4l2src device=/dev/video0 ! video/x-raw, width=640, height=480 ! videoconvert ! appsink
   * **Reason**: This pipeline reduces the resolution and frame rate, providing stable streaming without hangs.

### GStreamer Pipeline with Hardware Acceleration

* + **Pipeline**: nvarguscamerasrc ! video/x-raw(memory:NVMM), width=640, height=480, format=(string)NV12 ! nvvidconv ! video/x-raw, format=(string)BGRx ! videoconvert ! appsink
  + **Reason**: Utilizes NVIDIA’s hardware acceleration on ARM architecture for optimized video processing, ensuring smooth performance.

## Why Use GStreamer on Jetson Orin Nano?

* + **ARM Limitation**: High frame rates (60fps) cause instability due to limited processing power on ARM-based Jetson Orin Nano.
  + **OpenCV Limitation**: Direct camera access through OpenCV may result in high latency and frequent hangs.
  + **Stability and Performance**: GStreamer pipelines reduce resolution and leverage NVIDIA’s hardware acceleration, ensuring stable video streaming and efficient resource usage.
  + **Optimized for ARM**: GStreamer provides better handling of multimedia data, making it suitable for the ARM architecture of the Jetson Orin Nano.

## Additional Resources

Refer to the official NVIDIA Jetson Linux Developer Guide for detailed instructions on setting up GStreamer pipelines on Jetson devices: [Jetson Linux Developer Guide – GStreamer Pipelines](https://docs.nvidia.com/jetson/l4t/index.html)

## Conclusion

This GStreamer pipeline setup ensures smooth video streaming on Jetson Orin Nano, overcoming ARM architecture limitations by reducing frame rates and utilizing hardware acceleration, making it suitable for AI vision applications and real-time processing.

# Chapter 04: Integrating Camera and Ollama with ROS 2 Humble

This chapter describes the integration of a camera package and an Ollama-based LLM package within ROS 2 Humble. The camera publishes images to /camera/image\_raw, and the LLM package subscribes to this topic along with /user\_input, then publishes responses to

/llm\_response.

## Node Interaction Flow

→ **Camera Node** ➡ /camera/image\_raw ➡ **Ollama LLM Node**

→ **Ollama LLM Node** ➡ /user\_input ➡ **llava-7b model**

→ **lava-7b model** ➡ /llm\_response ➡ **Response**

## Key Points:

#### Camera Package:

* + - Publishes /camera/image\_raw using GStreamer pipeline for better ARM architecture support.

#### Ollama LLM Package:

* + - Subscribes to /camera/image\_raw and /user\_input.
    - Publishes LLM responses to /llm\_response.

This structure allows seamless real-time data flow between the camera feed with user inputs, and the LLM in a ROS 2 Humble environment.

## Results

The integrated system publishes images from the camera feed and generates responses from the Ollama LLM based on the visual input and user commands. The following results demonstrate the functionality:

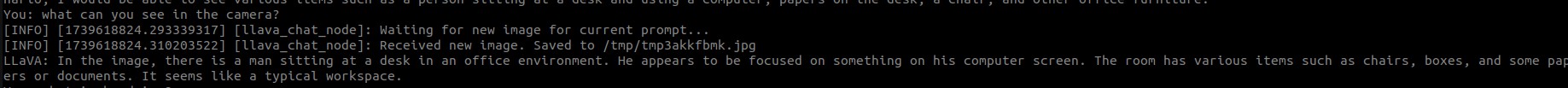
### Contextual Results with Images

#### General Conversation without image

* + **Normal Image**:

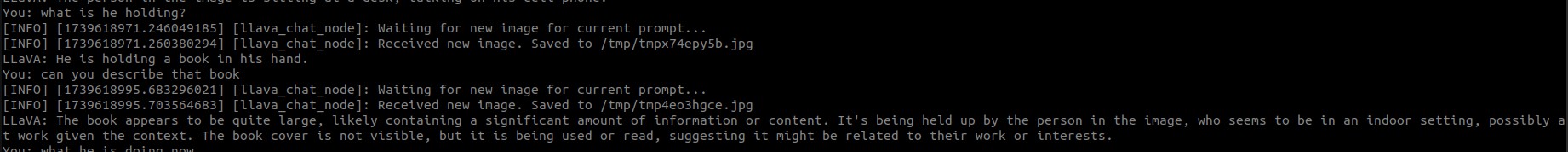


response =



#### Image with a Book:

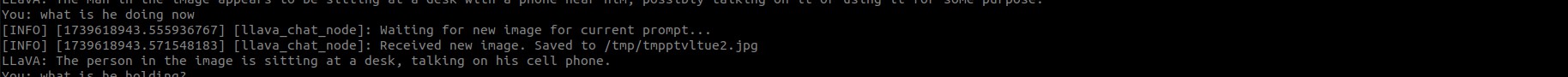


response =

#### Image with a Phone:



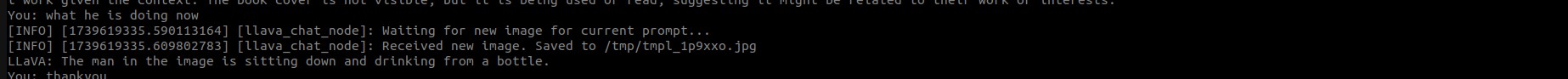
response =



#### Image with a Person Drinking Water:



response =



#### Conclusion

The integrated system successfully publishes images from the camera feed and generates accurate responses from the Ollama LLM based on visual input and user commands. The selected Llava-7b model has proven to be the right choice for the system, given its compatibility with the Jetson Orin Nano’s computational capabilities. This model will serve as the base for further development, including fine-tuning and integration with MoveIt for precise pick-and-place tasks as outlined in the project objectives.

This testing serves as a crucial foundation for future development by:

* + Validating the VLM model’s accuracy in interpreting visual data and user instructions for precise pick-and-place operations.
  + Establishing baseline performance with a webcam and identifying the need for a depth camera for more complex real-time operations.
  + Verifying seamless integration between Jetson Orin Nano, camera systems, ROS 2 Humble, MoveIt, and the Ollama LLM.
  + Demonstrating the system’s ability to process and respond to real-time inputs, essential for robotic manipulation tasks.
  + Providing a basis for further fine-tuning the VLM model to enhance its accuracy and functionality.
  + Confirming the operational feasibility of deploying VLM-based robotic solutions in real- world environments.
  + Identifying potential errors and areas for improvement, ensuring a robust final system.

This conclusion highlights that the current testing phase ensures a validated and reliable platform for future enhancements, including the integration of a depth camera and MoveIt for optimized pick-and-place operations, with the Llava-7b model as the foundation for all subsequent stages of development.

## Chapter 06: Selection of Depth Camera

*(Content to be added)*

## Chapter 07: Model Fine Tuning

*(Content to be added)*

## Chapter 08:

*(Content to be added)*