Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Fachbereich Informatik Kognitive Systeme

Bachelor Thesis Computer Science

Building a ROS2-VLA Bridge

Samuel Rochlitzer

30.10.2025

Reviewer

Prof. Andreas Zell
Department of Computer Science
University of Tübingen

Supervisor

David Ott
Address
University of Tübingen

Name, first name: Rochlitzer Samuel
Building a ROS2-VLA Bridge
Bachelor Thesis Computer Sciences
Eberhard Karls Universität Tübingen
Period: 01.06.25 - 31.10.25

Abstract

This bachelor thesis investigates the integration of vision-language-action (VLA) model into the robot operating system 2 (ROS2) framework for robotic control. To enable this, a modular bridge was developed that separates the ROS2 client package from a Python backend running the VLA. Communication between the two components is realized through a lightweight *Flask* server. The system is containerized via Docker for reproducible deployment on a *Franka Emika Panda* arm, while a dedicated *conda* environment supports execution of graphics processing unit (GPU) intense vision-language-action (VLA) models. The bridge node collects image data, joint states, and end-effector poses, combines them with the textual prompt, and forwards this information to the backend. The predicted motion commands are then mapped to a *cartesian impedance controller* to execute actions on the robot. The interface was validated with *OpenVLA*, demonstrating end-to-end communication and successful command execution, albeit with limited performance in this environment. The results suggest that integration is feasible, but highlight the need for models fine-tuned to the specific robot setup in order to achieve reliable task execution.

Acknowledgements

Of course, I am building on the work of my research group with Prof. Andreas Zell and my supervisor David Ott.
I also want to thank my brother for listening, my girlfriend for the support and my friends who provided feedback on style, structure and flow.

Contents

List of Figures

List of Tables

List of Abbreviations

LLM	large language model
VLM	vision-language model
VLA	vision-language-action
ROS2	robot operating system 2
SPA	sense-plan-act
DDS	data distribution service
GPU	graphics processing unit

Chapter 1

Introduction

"...it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility..." [Mor95]

In this apparent paradox, Hans Moravec captured one of robotics' most persistent observations from the 1980s till today. As computers surpass humans at tasks of higher reasoning -like math or chess- we still struggle to automate some of the unconscious tasks, we perform effortlessly.

Forty years later, advances in artificial intelligence reopen questions. Can machines now connect perception, language, and motion? Should we adopt entirely new paradigms, or rather integrate these models within established robotic principles?

This thesis investigates how such models can be integrated into existing robotic frameworks through a ROS2–VLA bridge. The introduction first motivates and defines this goal, before outlining the central questions and the approach used to address them.

1.1 Motivation

Since the early days of robotics, researchers have observed that tasks trivial for humans often demand complex control strategies and a deep understanding of objects and their spatial relations. Transformer-based large language model (LLM)s have made significant progress in modeling such relations from natural language [Dev+19]. When extended with a vision encoder, these models become vision-language model (VLM)s capable of interpreting both images and text [Liu+23]. Fine-tuning a VLM on robot trajectory data yields a vision-language-action (VLA) model that can generate actions based on the robot's state, visual input, and natural-language instructions [Bro+23b; Kim+24]. Since their emergence around 2021 [SMF21], VLAs have demonstrated increasingly generalizable reasoning and control abilities, further expanding their potential [Sha+25].

Despite this promise, VLA models mainly produce low-level motion increments—typically pose or joint deltas—that must be applied continuously to the robot. Their non-deterministic behavior and black-box nature make integration with existing control frameworks challenging.

To leverage these models in real robotic systems, they must be connected to established middleware. The robot operating system 2 (ROS2) framework provides modularity, safety mechanisms, and visualization tools, making it a natural integration target. Building a bridge between VLA models and ROS2 would enable their combination with traditional components and allow deployment in more complex robotic systems.

1.2 Problem Statement

Modern VLA models exhibit impressive task flexibility, but their outputs remain limited to incremental deltas without guarantees of stability or safety [Bro+23b; Kim+24]. Integrating such models into robotics frameworks like ROS2 is further hindered by differing software dependencies: VLA models require GPU-intensive machine-learning stacks [Vas+17], whereas ROS2 relies on lightweight middleware and driver integration [Mac+22].

At the same time, the computational profile of VLA models—with potentially high inference latency—contrasts sharply with the real-time requirements of robotic controllers [Fra]. This gap motivates the development of a dedicated bridge that can reconcile these mismatches and enable reliable deployment within established robotic systems.

1.3 Research Questions and Evaluation Plan

The central research question of this thesis is how a modular interface can be designed to integrate VLA models with the ROS2 control framework while maintaining stability, modularity, and low latency. This question is investigated through the design and implementation of a client–server bridge architecture that separates model inference from robotic control.

The work explores how such a bridge can reconcile the asynchronous, GPU-intensive inference of VLA models with the timing and safety constraints of ROS2 controllers [Vas+17; Mac+22]. Beyond conceptual design, the study examines how the architecture performs under realistic workloads in terms of latency, success rate, and control stability, and how its modular design allows the substitution of different backends or robotic setups with minimal adaptation.

The evaluation follows an iterative, experiment-driven approach combining real-world testing on a Franka Emika Panda platform with quantitative performance analysis [Fra]. Two experiment categories were conducted: baseline verification with deterministic hard-coded deltas to validate the full communication and control pipeline, and model-based inference runs with just *OpenVLA-7B* and using locally trained checkpoints [Kim+24]. Each experiment logged timestamps, pose and meta data, which were evaluated using a custom script analyzed in chapter 5.

The evaluation aims to verify whether the bridge maintains stable communication between the robot and backend, ensures consistent control behavior despite variable inference times, and can recover gracefully from delays or backend errors. Since the objective is to enable integration of VLA models into the ROS2 framework, the thesis does not benchmark or optimize specific models. Instead, it demonstrates the feasibility of the proposed architecture, identifies performance bottlenecks, and provides a reference design for VLA–ROS2 integration.

1.4 Thesis Structure

To help maintain a explanatory structure we decided to keep the Intro in chapter 1 short and dedicate Context and Theory it's own chapter 2 explaining fundamentals before the chosen approach, scope and assumptions of this project. For our own work the thesis covers hardware, software, and the bridge architecture in chapter 3 Implementation and methods behind it and it's evaluation in chapter 4. Next chapter 5 is about the recordings and experimental results using the software, before chapter 6 discusses the findings reflecting on limitations, broader context and future Work.

We make all code and records available at:
`https://github.com/aZamuel/ROS2-VLA_Bridge.git`

Chapter 2

Fundamentals and Concept

Before presenting the bridge architecture, this section outlines the theoretical foundations that inform its design. It begins with established control paradigms in robotics and the evolution toward layered architectures, followed by the role of ROS2 as a modern middleware. Finally, it traces the development of VLA models from large language models and discusses their relevance to robotic control.

2.1 Conventional Robotics

Modern robotics builds on decades of work in control theory [Gat98]. Early systems followed the sense-plan-act (SPA) paradigm: the robot first perceived its environment, then computed a plan based on an internal model, and finally executed that plan through motor control. This approach provided clear modularity and allowed verifiable reasoning but proved slow and brittle in changing environments. In response, *reactive* or *behaviour-based* control emerged, most notably in Brooks's *subsumption architecture*, which organized behaviour into layered, sensor-driven modules capable of suppressing lower-priority ones. This style improved robustness and timing at the cost of explicit planning.

The prevailing compromise today is the *hybrid three-layer architecture*, integrating deliberative planning at the top, sequencing in the middle, and fast reactive control at the bottom. This structure underlies most practical robot control frameworks, including those implemented in ROS2, emphasizing modularity, synchronization, and stable feedback loops.

2.2 ROS and ROS2

The robotics community required a framework to connect these layers efficiently, leading to the development of the robot operating system (ROS) around 2010 by Willow Garage. ROS provided a distributed message-passing architecture with topics, services, and actions, enabling reusable components across research and industry. Its limitations—centralized master node, lack of real-time support, and difficulty scaling to multi-robot systems—motivated a fundamental redesign.

ROS2, first released in 2017, addressed these constraints. Built on the data distribution service (DDS) standard, ROS2 introduced configurable quality of service (QoS) profiles for reliability, deadlines, and message persistence, making it suitable for distributed and time-critical control. It also added deterministic communication, improved security, and cross-platform compatibility.

This thesis uses the *Humble Hawksbill* distribution, offering a stable baseline for integration. ROS2's dual client libraries—C++ (`rclcpp`) and Python (`rclpy`)—allow developers to choose between performance and flexibility. Python support is especially relevant here, as it enables rapid prototyping and easier integration with machine-learning frameworks although we stuck to a pure *c-make* build as it is more stable. [**ros2docs**; Mac+22]

2.3 From LLM to VLA

The emergence of large language models (LLMs) fundamentally changed how artificial systems represent and manipulate knowledge. Based on the Transformer architecture [Vas+17], these models learn contextual relations across massive text corpora and demonstrate reasoning capabilities beyond symbolic approaches [Dev+19].

Extending these methods to the visual domain produced vision-language model (VLM) models that ground linguistic understanding in perception [Liu+23]. Models such as CLIP [SMF21] align images and captions in a shared embedding space for zero-shot recognition, while others can interpret sequences of interleaved text and images for multimodal reasoning.

VLA models represent the next step towards robotics. They add an action head that maps visual and language inputs to continuous motor commands. Typically, a VLA consists of a visual encoder, a language-conditioned transformer core, and an action decoder producing incremental 6-DoF motions and a gripper signal. [Bro+23a]

VLA models are trained on large-scale robot-demonstration datasets such as BridgeData V2 and Open X Embodiment [ONe+25], which pair camera observations, language descriptions, and corresponding actions [ONe+25].

Well-known examples include RT-2 [Bro+23b], OpenVLA [Kim+24] and $\pi0$ [Bla+24]. While VLAs enable instruction-driven manipulation and semantic generalization, they operate reactively, predicting short incremental deltas rather than full trajectories. Inference is GPU-bound and introduces variable latency; models have limited temporal memory and embodiment awareness [Sha+25]. These properties contrast sharply with the deterministic, fixed-rate expectations of ROS2 control systems and motivate the development of a dedicated bridge between both domains.

2.4 Proposed Bridge

The proposed bridge follows a client–server design. On the ROS2 side, a dedicated package provides a client node that manages communication with an external backend. The node exposes service interfaces to start, stop, or configure a request loop and maintains the current sensor context, including image and joint state data. It periodically forwards this context to the backend, consisting of a lightweight server and a wrapper. When started, it instantiates a VLA model and listens for requests. Upon receiving a request, it prepares the input data in the required format for the model and returns action deltas predicted. (s. chapter 3 for more)

This architecture decouples the machine learning environment from the robotics middleware, which ensures modularity and keeps the ROS2 side lightweight. In principle, the backend could also run on a separate system with dedicated GPU resources, or support different VLA models with minor changes. (s. chapter 4)

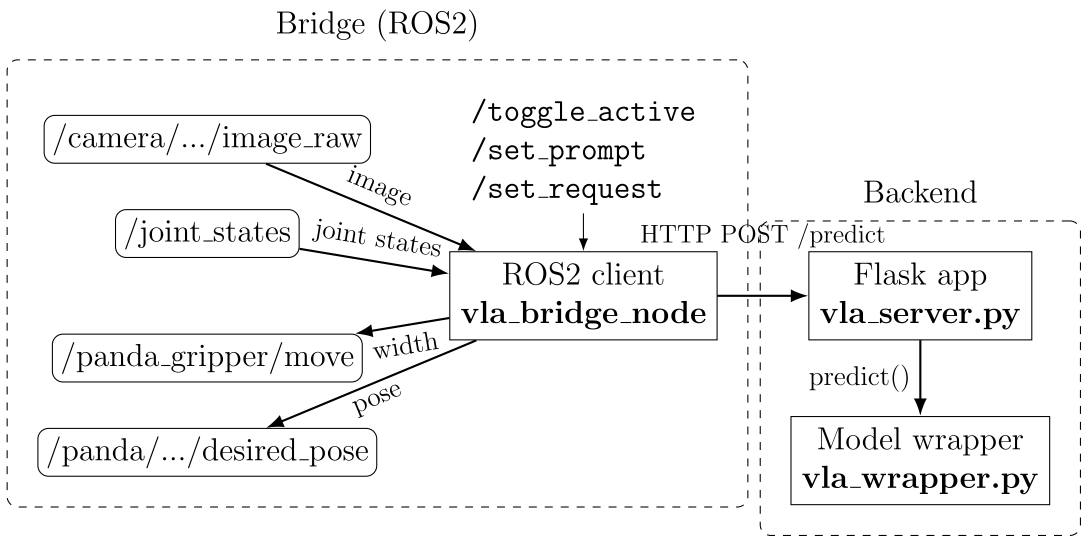


Figure 2.1: Architecture of the ROS2-VLA Bridge

2.5 Contributions

The main contributions of this thesis can be summarized as follows:

- Development of a ROS 2 client node 3.3.2 (**vla_bridge_node.py**) that connects camera and joint state topics to a backend hosting a VLA, and publishes Cartesian impedance goals to the Franka controller.
- Implementation of a backend 3.3.1 consisting of a Flask application (**vla_server.py**) and a wrapper interface (**vla_wrapper.py**), enabling the integration of different VLA models through a unified REST API.
- Design of an interfaces package for runtime control, including services **/toggle_active**, **/set_prompt** and **/set_request**.
- Provision of a reproducible deployment environment by means of a **Dockerfile** and a **environment.yml** specification.

2.6 Scope, Assumptions, and Constraints

The scope of this thesis is limited to developing a bridge that enables the use of Vision-Language-Action models within ROS2. The focus lies on the ROS2 client node and its interfaces, while the backend is only tested with *OpenVLA*. Further fine-tuning or developing new VLA models is explicitly out of scope.

The work assumes that the backend provides action deltas at a usable rate and that sensor data from camera and joint states is consistently available. It further assumes that these deltas can be reliably converted into valid absolute pose targets for the *cartesian impedance controller*.

Constraints arise from the dependency gap between robotics middleware and machine learning stacks, which are isolated by containerization on the ROS2 side. Performance is limited by the inference speed of the chosen VLA model and by the strict requirement of the Cartesian controller to only accept absolute poses within a small step size of $0.1m$ [MS22]. Safety and monitoring are recognized as important aspects in robotics, and dedicated ROS2 packages for these purposes exist, but their integration is not part of this thesis.

Chapter 3

Implementation

In this chapter implementation of the ROS2-VLA Bridge is described. The aim is to outline the system components and their interactions used to connect VLA models with the ROS2 control framework. The following sections provide an overview of the overall architecture, the backend implementation, the ROS2 integration, and deployment.

3.1 System Overview

A Flask-based backend runs the model and provides a **/predict** endpoint, while a python wrapper handles model loading and inference. On the ROS2 side, a client node manages communication: it captures camera images and joint states, converts them into a request to the backend, and publishes the resulting pose to the *cartesian_impedance_controller*. Interaction with the bridge node is handled through ROS2 services (`/toggle_active`, `/set_prompt` and `/set_request`).

3.2 Hardware Setup

The hardware platform consists of a Franka Emika Panda robot arm with seven degrees of freedom. Control is provided through the Panda's Cartesian impedance controller, which accepts target poses on the `/panda/panda_cartesian_impedance_controller/desired_pose` topic. The reference frame for end-effector control is defined at `panda_hand_tcp`. Visual input is supplied by an Intel RealSense camera, which streams image data into the ROS2 system. The robot is connected to the host workstation via Ethernet, while the workstation itself runs Ubuntu 22.04 with a `NVIDIA® GeForce RTX™ 5090` GPU to support model inference.

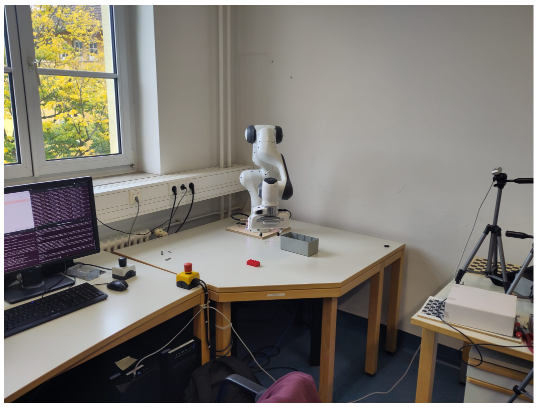


Figure 3.1: Franka Emika Panda with Realsense camera

3.3 Software Stack

The software stack combines a Dockerized ROS2 environment with a Conda-based backend. The system runs on Ubuntu 22.04 with ROS2 Humble, where two custom packages were developed: one hosting the bridge node and another providing service interface definitions. ROS2 itself is run in a Docker container, ensuring consistent conditions across systems. The machine learning backend is kept separate from this container and runs in a Conda environment specified by **environment.yml**. It is divided into two components: a lightweight Flask server that exposes predictions through a **/predict** route, and a wrapper module responsible for loading the OpenVLA model via *HuggingFace Transformers*. Dependencies such as PyTorch are installed in the Conda environment. *FlashAttention_2* is setup to be enabled automatically when available, the software should optimize GPU performance, but could not be installed at the time due to dependency conflicts. The initial ROS2 configuration, based on a *multipanda_ros2* Docker setup provided by my supervisor, was extended and adapted to support client integration.

3.3.1 Backend Integration

The backend provides the inference service that connects ROS2 to the VLA model. It is organized into two components. The first is a Flask app (**vla_server.py**) that exposes a HTTP route(**/predict**) to decode and handle incoming requests. The second is a wrapper module (**vla_wrapper.py**) that loads the *OpenVLA* model through *HuggingFace* and performs the actual inference.

Requests from the ROS2 client contain four elements: a *base64* encoded image from the camera, the current joint configuration, a textual prompt describing the intended action, and the model name for future proofing. The images are decoded, converted to RGB, resized to 224×224 pixels, and passed to the model together with the prompt, which is inserted into a standard instruction template as described in the OpenVLA framework. The output follows the OpenVLA action format, consisting of Cartesian translations (Δx, Δy, Δz), rotations ($\Delta roll$, $\Delta pitch$, Δyaw), and a gripper width value. This structure is consistent with *BridgeData V2* conventions and can later be applied to the current pose for the controller.

Dependencies like machine learning libraries are specified via a conda environment file. The separation between server and wrapper increases modularity and makes it straightforward to substitute different VLA models in the future.

3.3.2 ROS2 Integration

ROS2 integration is realized by two custom packages: **vla_interfaces** and **vla_client**. The first defines the service types, while the second implements the bridge node that connects the ROS2 control stack with the backend. The bridge node subscribes to the RealSense camera and joint states and publishes Cartesian impedance goals to the Panda controller.

The node exposes three services. **/toggle_active** starts or stops the request loop. **/set_prompt** updates the natural-language instruction used for prediction. It accepts a string as request and responds with a success flag and a message. The third, **/set_request**, allows reconfiguration of backend-related parameters such as the request interval, the backend URL, model name and active flag all at once. This modular design makes it possible to switch between different models without changing the ROS2 client packages.

When active, the node periodically collects the latest image, joint configuration, the current prompt, and the model name. These values form a request that is sent to the backend, which returns predicted action deltas and a target gripper width. Including the model name ensures future-proofing, as the same client can address different models without modification.

On the client side, the response is processed step by step. First, the translational and rotational deltas are saturated to limit their magnitude per control cycle. Next, the current end-effector pose is retrieved via TF2 transformations between *panda_link0* and *panda_hand_tcp*. The saturated deltas are then added to the position of the current pose, while orientation updates are composed using quaternion multiplication [MKW25]. The final result is a Cartesian target pose, which is published on the Panda controller topic as the desired goal. This sequence implements the typical control pipeline described in Cartesian impedance examples [MS22] while adapting it to the action format produced by OpenVLA [Kim+24].

3.4 Deployment

The system is deployed in two environments: a Docker container for ROS2 and a Conda environment for the backend. The ROS2 container is built from a multipanda_ros2 base image and extended with, among others, the custom `vla_client` and `vla_interfaces` packages. After launching the container, the robot arm and gripper driver and the RealSense camera node are started, followed by the bridge node.

The backend runs independently in a Conda environment defined by the `environment.yml` file. After activating the environment, the Flask server is launched to host the OpenVLA model. Communication between ROS2 and the backend occurs over HTTP through the `/predict` endpoint.

In practice, deployment involves the following steps (once):

1. (Build and) start the ROS2 container.
2. Launch the required nodes manually or using **launch_all**.
3. (Create and) activate the Conda environment.
4. Start the backend **vla_server**.
5. Configure and activate the bridge node using the **/set_request** service.

Chapter 4

Methods

This chapter explains the methodological reasoning behind the ROS2–VLA Bridge: (4.1) Design principles and engineering choices that make the bridge modular, re-usable, and compatible; and (4.2) Experimental Setup, data collection, and metrics used to evaluate latency, stability, and control behavior on a Franka Panda platform.

4.1 Design Principles

The bridge was designed as a **decoupled client–server system** to reconcile the different computational and timing requirements of robot control and large-scale model inference. Separating the ROS2 control domain from the VLA backend enables the inference process to run on dedicated GPU hardware, while the robot host remains as lightweight as ROS2 would allow [ref:rosbridge]. Communication between the two follows a minimal JSON contract transmitting only the current image, joint configuration, instruction prompt, and model identifier. This compact data format ensures compatibility with future VLAs backends and keeps communication overhead to a minimum [Kim+24].

To maintain **modularity and model independence**, all VLA-specific logic is encapsulated inside the **vla_wrapper**. It standardizes preprocessing, prompt formatting, and inference calls, so the ROS2 side can remain unchanged even when different model checkpoints or architectures are used. This abstraction also defines conservative fallback behavior: if decoding or inference fails, the backend returns zero-motion deltas, allowing the bridge to fail safely without disrupting the control loop [Fra].

The **vla_interface** exposes three services for runtime configuration—activation, prompt change, and backend parameter update—while subscribing to camera and joint-state topics and publishing Cartesian pose goals. This service-oriented control structure enables on-the-fly experimentation without redeployment and follows the modular runtime interaction principles recommended for mixed human-in-the-loop systems [Mac+22].

Complementary to runtime configuration, the bridge also supports **static node and topic parameterization** through a dedicated *profile.yaml* configuration loaded in the **launch_all.launch** file. This YAML profile defines topic names, Quality-of-Service parameters, and transform frames, allowing adaptation to different ROS2 setups such as alternative camera drivers or controller namespaces. Unlike the dynamic services, these parameters require a node restart to take effect, but they provide a robust mechanism for reusing the bridge across diverse environments without modifying source code [Mac+22].

Design choices within the ROS2 domain were guided by the goal of **preserving timing stability and data integrity** rather than achieving hard real-time performance. Reliable Quality-of-Service profiles are used for sensor topics to prevent packet loss, and each request loop is timestamped to monitor latency between message publication and backend response. This best-effort synchronization follows established practices for soft real-time systems in ROS2 [Mac+22].

Consistency across coordinate frames is ensured through TF2, which maintains the transform between *panda_link0* and *panda_hand_tcp*. All predicted deltas are saturated and composed into new absolute poses that respect the Cartesian impedance controller's motion limits, guaranteeing stable interaction with the physical robot [Fra].

Safety and fault tolerance are integral parts of the design. Gripper actions are scaled, gated and stopped to prevent overlap, and the bridge enforces bounded step sizes per cycle. When sensor data or backend responses are invalid, the system defaults to no-op behavior and logs the event. These mechanisms proved essential during experimentation to avoid cascading control faults.

Finally, the architecture emphasizes **reproducibility and portability**. The ROS2 workspace is containerized via Docker, while the backend operates in an isolated Conda environment pinned to the specific versions. This separation enables deployment across machines and supports distributed setups where the backend runs remotely on a GPU cluster (s. chap 6).

4.2 Experimental Setup

This section outlines the experimental methodology used to validate the ROS2–VLA Bridge. It describes the physical setup, the configuration of hardware and software components, and the procedure used for recording and analyzing data. All experiments were conducted under consistent environmental conditions to ensure comparability across runs.

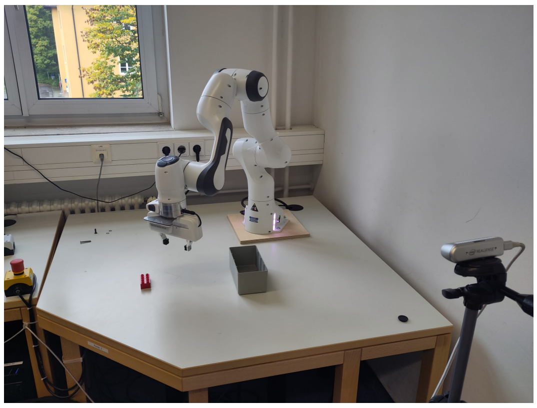


Figure 4.1: Hardware setup (right perspective)

4.2.1 Hardware Configuration

Experiments were performed on a workstation equipped with an NVIDIA RTX 5090 GPU running Ubuntu 22.04. The robot platform consisted of a Franka Emika Panda arm equipped with a Cartesian impedance controller and an Intel RealSense camera. The camera was mounted on a tripod to maintain a stable perspective on the workspace, which contained two objects used in all experiments: a red Duplo brick and a gray box positioned near the center of the robot's reachable area. The robot's end-effector started from a neutral position between the two objects at a height chosen for unobstructed motion. Lighting and surrounding conditions were kept constant across all recordings.

4.2.2 Software Configuration

The ROS2 stack was executed inside a Docker container containing the workspace with the *Franka Emika driver* and the *realsense2_camera* nodes. The backend ran separately in a Conda environment on the same host system to isolate dependencies. Inside this environment, a **vla_server** instantiated the **vla_wrapper**, loading the *openvla/openvla-7b* model from Hugging Face. Both components communicated locally over HTTP through the */predict* endpoint, ensuring minimal latency and a controlled communication context. At the time of development, no FlashAttention 2 binary was available for the specific CUDA/GPU combination, which may have limited achievable inference speed.

Within the container, the **vla_bridge_node** subscribed to the image and joint-state topics and published Cartesian pose goals to the */panda/panda_cartesian_impedance_controller/desired_pose* topic, as well as gripper width commands to the */panda_gripper/move* action. Runtime control and reconfiguration were managed through the */set_request* and */toggle_active* services.

Static parameters such as topic names, frame identifiers, and Quality-of-Service settings were provided through a dedicated YAML configuration file loaded by **launch_all.launch**. This separation between runtime and launch-time configuration enabled quick experimental iteration while preserving reproducibility across sessions.

4.2.3 Recording Procedure

For each run, the ROS2 node recorded timing information, end-effector positions, orientations, and gripper widths to a CSV file. Each experiment consisted of at least 50 request cycles, which were considered the evaluation window for all quantitative metrics. Requests were executed at 1 Hz, 2 Hz, and 5 Hz to analyze the effect of update frequency on bridge stability and controller behavior. Backend configuration, active prompt, and camera perspective were varied systematically, while all other parameters remained constant. Recordings and scripts are provided as supplementary material.

4.2.4 Evaluation Script

Data analysis was performed using the **eval.py** script developed for this project. It processes the recorded CSV files, extracts the first 50 entries, and computes performance metrics such as end-to-end latency, backend inference time, success rate, and total Cartesian and rotational motion. In addition to numerical summaries, the script generates three-dimensional trajectory and angle plots for each recording, providing visual insight into controller behavior and timing stability. This automated evaluation pipeline ensured consistent post-processing across all experiments.

4.2.5 Experiment Series

Two experiment categories were performed:

Hard-coded

To isolate the bridge from model-specific behavior, the backend was configured to bypass inference and return fixed deltas for command marked **"HARD: ..."**. For those prompts, the backend was programmed to return:

$$\texttt{results} = \begin{cases} \texttt{delta\_z} = \pm 0.01m, & \text{for go up/down} \\ \texttt{delta\_*} = \pm 0.5 \text{ rad}, & \text{for * rotation} \\ \texttt{delta\_gripper} = \pm 0.05m, & \text{for open/close gripper} \end{cases}$$

while all other deltas were set to zero. Two request frequencies were compared: 1 Hz, where each movement finished before the next request, and 5 Hz, where updates overlapped for near-continuous motion. These runs verified end-to-end communication, pose update handling, and gripper synchronization at 1 Hz and 5 Hz.

Model Inference

The same setup and configuration described in Section 4.2 were used throughout. Two backend configurations were compared: a locally fine-tuned model for a specific task and the base *OpenVLA-7b* [Kim+24] model without any adaptation. Each configuration was evaluated with request frequencies of 1, 2, and 5 Hz using two camera perspectives.

Finetuning attempt: To test whether local fine-tuning could improve model performance, data for the task *"put duplo brick in the box"* were recorded and trained on the *Avalon1* GPU cluster. The resulting weights were integrated into the backend by specifying a local directory as in *deploy.py* and updating the corresponding *unnorm_key.* The evaluation followed the same procedure as before, running the bridge at 1 Hz, 2 Hz, and 5 Hz from two different camera positions.

Base model: For comparison, the original *OpenVLA-7b* model [Kim+24] was used without any local fine-tuning. The backend was modified to deactivate the loading of local files, forcing the system to run the pretrained weights directly from the Hugging Face repository. The same *"put duplo brick in the box"* prompt and identical recording setup were used, and in addition, the *"pick up the red cube"* example prompt from the *OpenVLA* documentation was tested at 2 Hz.

Chapter 5

Results

This chapter addresses the question of how the bridge performed in the experiments, allowing integration of VLA models with minimal adjustments. The following section will describe validation runs with hard-coded responses and evaluations with actual VLA models. We will also expand on the timings recorded and the overhead involved of adapting the bridge to a new VLA possibly on a different system.

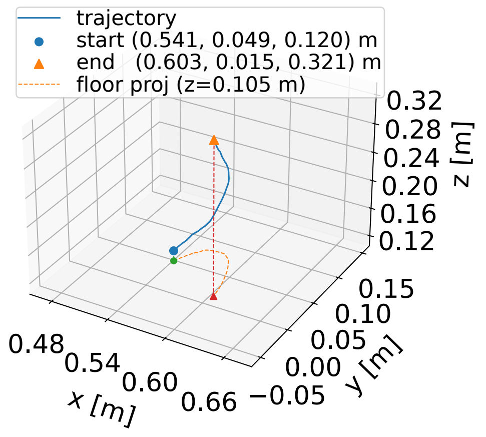
5.1 Hard-coded

To verify basic functionality like communication and the control stack behavior we started with prompts hard-coded to return consistent deltas.

Even under these controlled conditions, several issues became apparent. In the translation runs, the robot displayed drift in directions where zero deltas were applied, and the total displacement along the commanded axis was smaller than expected. For example, applying a **delta_z** of 0.01 m over 50 requests resulted in only 0.20 m total motion at 1 Hz and 0.11 m at 5 Hz. Adjusting for the wrist-mounted camera mass on the end-effector did not noticeably change this behavior.

The rotation tests produced mixed results. Pitch and roll both caused small translations in addition to the expected angular motion. The roll motion occasionally stopped completely partway through a sequence, while yaw remained the most stable and isolated. These rotational issues were independent of the request frequency and likely stem from the way small deltas are accumulated into a new pose computed from the current joint configuration before publishing to the *cartesian impedance controller*.

The gripper movements further exposed overlap problems. At 1 Hz, open and close actions completed cleanly. As the gripper is moved with an action instead of just publishing a goal to a topic it also logs success. At 5 Hz, new



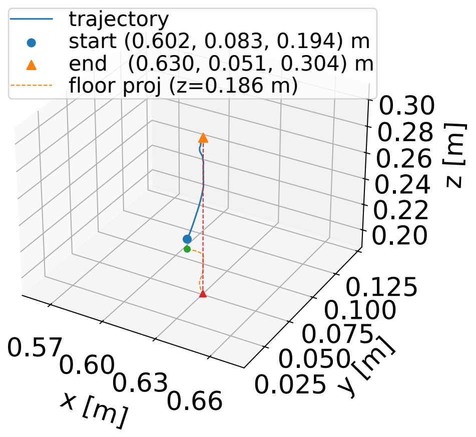
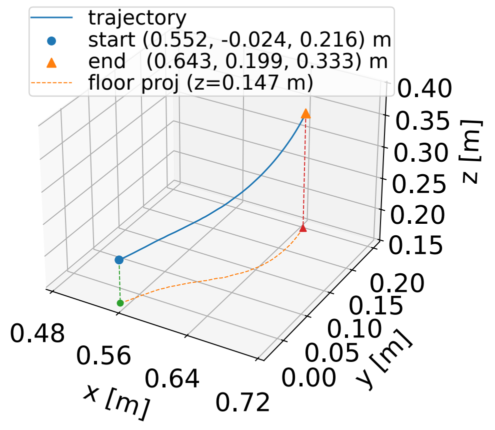


Figure 5.1: Trajectories with "HARD: go up" at 1 Hz and 5 Hz

move commands were issued before the previous actions had finished, leading to stacked or delayed executions. In some runs, the gripper continued moving even after the bridge had been deactivated.

To mitigate these issues, stop commands were added between consecutive gripper actions, and the motion step size was scaled with the request interval to ensure physical feasibility. Calling to frequent then resulted in the gripper not moving at all. Finally movements smaller than one millimeter were ignored to reduce lag buildup.

These tests confirmed that communication between the backend and the ROS2 node worked reliably and that the bridge could publish goals to the controller. However, they also revealed major synchronization and accumulation



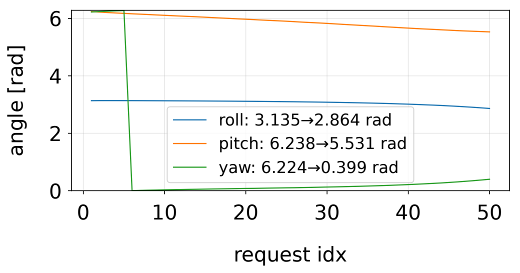
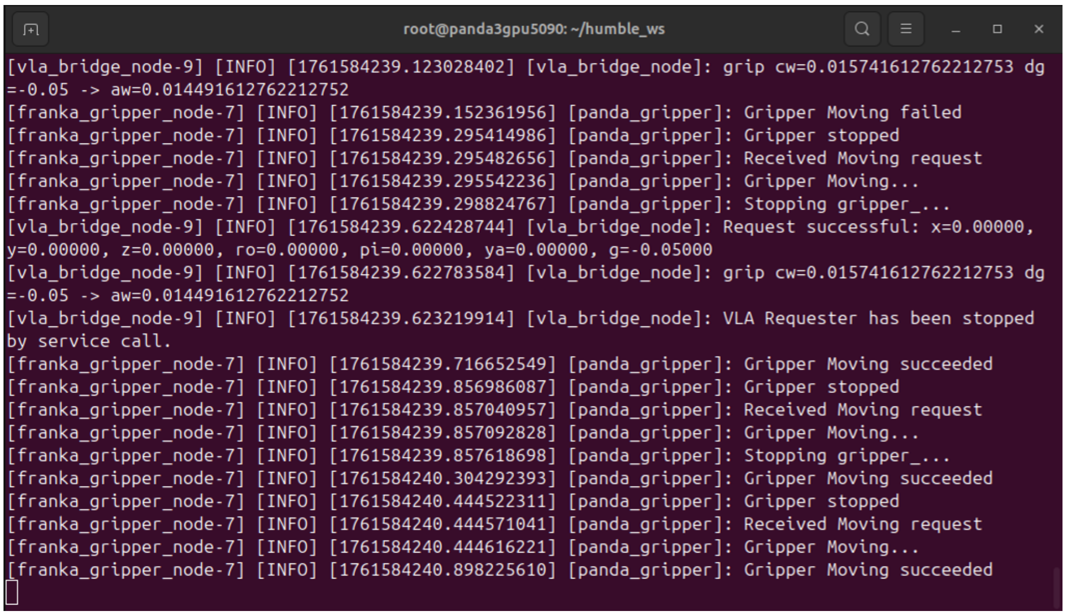


Figure 5.2: Trajectory and rotation with "HARD: pitch up" at 1 Hz



```
[vla_bridge_node-9] [INFO] [1761584239.123028402] [vla_bridge_node]: grip cw=0.015741612762212753 dg
=-0.05 -> aw=0.014491612762212752
[franka_gripper_node-7] [INFO] [1761584239.152361956] [panda_gripper]: Gripper Moving failed
[franka_gripper_node-7] [INFO] [1761584239.295414986] [panda_gripper]: Gripper stopped
[franka_gripper_node-7] [INFO] [1761584239.295482656] [panda_gripper]: Received Moving request
[franka_gripper_node-7] [INFO] [1761584239.295542236] [panda_gripper]: Gripper Moving...
[franka_gripper_node-7] [INFO] [1761584239.298824767] [panda_gripper]: Stopping gripper_...
[vla_bridge_node-9] [INFO] [1761584239.622428744] [vla_bridge_node]: Request successful: x=0.00000,
y=0.00000, z=0.00000, ro=0.00000, pi=0.00000, ya=0.00000, g=-0.05000
[vla_bridge_node-9] [INFO] [1761584239.622783584] [vla_bridge_node]: grip cw=0.015741612762212753 dg
=-0.05 -> aw=0.014491612762212752
[vla_bridge_node-9] [INFO] [1761584239.623219914] [vla_bridge_node]: VLA Requester has been stopped
by service call.
[franka_gripper_node-7] [INFO] [1761584239.716652549] [panda_gripper]: Gripper Moving succeeded
[franka_gripper_node-7] [INFO] [1761584239.856986087] [panda_gripper]: Gripper stopped
[franka_gripper_node-7] [INFO] [1761584239.857040957] [panda_gripper]: Received Moving request
[franka_gripper_node-7] [INFO] [1761584239.857092828] [panda_gripper]: Gripper Moving...
[franka_gripper_node-7] [INFO] [1761584239.857618698] [panda_gripper]: Stopping gripper_...
[franka_gripper_node-7] [INFO] [1761584240.304292393] [panda_gripper]: Gripper Moving succeeded
[franka_gripper_node-7] [INFO] [1761584240.444522311] [panda_gripper]: Gripper stopped
[franka_gripper_node-7] [INFO] [1761584240.444571041] [panda_gripper]: Received Moving request
[franka_gripper_node-7] [INFO] [1761584240.444616221] [panda_gripper]: Gripper Moving...
[franka_gripper_node-7] [INFO] [1761584240.898225610] [panda_gripper]: Gripper Moving succeeded
```

Figure 5.3: Logs while closing the gripper (at deactivation)

problems within the control stack, especially when small deltas were repeatedly added to the current pose calculated. These findings provided a baseline for later model-based experiments but also highlighted key limitations discussed further in Chapter 6.

5.2 Model Evaluation

As described in Methods (4.2.5) we recorded some representative runs using our, limited, setting- and task-specific, training data. The statistics produced from our minimal finetuning attempt where integrated to the best of our abilities, according to OpenVLA documentation (*deploy.py*) [Kim+24]. We also compare the results with the base model experiments on the same task and it's default prompt.

Note, that many different variations of setup, camera perspectives and prompts where tested through out development. As they didn't yield meaningfully different results, we discuss the results using these representative recordings.

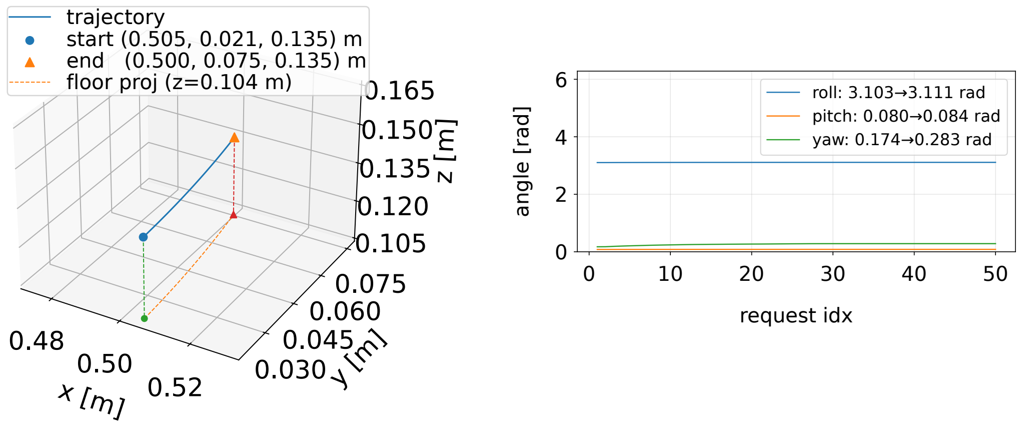


Figure 5.4: Trajectory and rotation on finetuned task 2 Hz

5.2.1 Finetuning attempt

The fine-tuned model, did not produce the intended behavior. In all runs, the robot repeatedly moved toward a "neutral" position around $x \approx 0.5$ m and $y \approx 0$ m, without changing height, orientation, or gripper width. The movement gradually slowed until nearly static, while the backend continued to return only minuscule deltas. Communication between the backend and the ROS2 node remained stable and error-free, indicating that the issue did not originate from the bridge itself but from the model output. The results suggest a form of output freezing, possibly due to normalization mismatch or insufficient variation in the fine-tuning data, though the exact cause could not be confirmed.

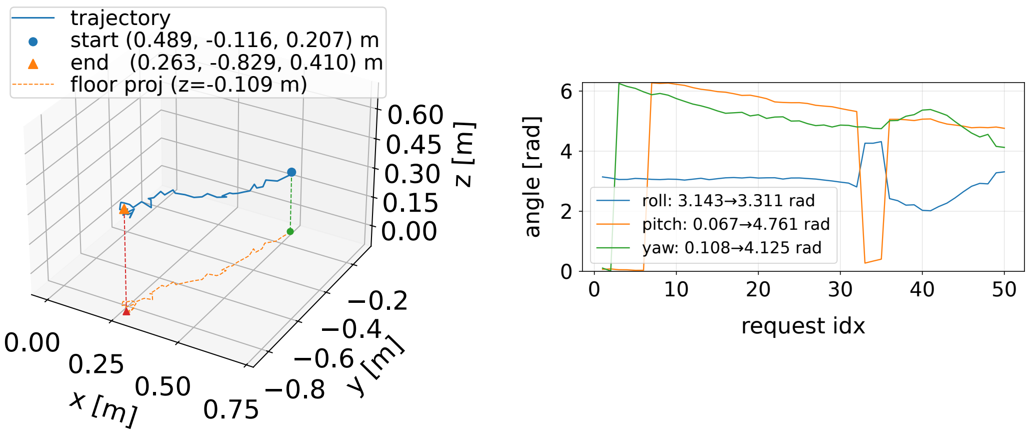


Figure 5.5: Trajectory and angles of base model on default task at 2 Hz

5.2.2 Base Model

With the base model, the robot showed continuous but unstructured movement. It moved in varying directions across the workspace, but not toward the target objects. The trajectory changed slightly between camera perspectives but showed no consistent improvement. Unlike the fine-tuned version, the model did not freeze and produced persistent small deltas, yet these remained semantically unrelated to the task. This behavior reveals the model's limited generalization across different physical settings.

Both sets of experiments confirm that the bridge successfully transmits real inference results through the ROS2 control stack, with consistent timing and data integrity. However, neither model variant achieved task-directed motion. While the fine-tuned model converged to static outputs, the base model generated random yet continuous movements, highlighting both the robustness of the bridge implementation and the shortcomings of the current model integration. Quantitative timing data for these runs are presented in the next section.

5.3 Latency

This section examines the timing behavior of the bridge during operation. The measurements distinguish between two main components: the communication overhead between the ROS2 client and the backend, and the time required by the VLA model to generate an action prediction.

Hard-coded Tests (Communication Overhead)

The hard-coded runs provide a baseline for message transfer performance. For all translation, rotation, and gripper tests, end-to-end processing times remained below 10 ms. Most of this delay resulted from encoding the camera image to *base64*, sending the HTTP request, and decoding the reply before calling **predict()** in the **vla_wrapper**. This represents the bridge's communication overhead.

Table 5.1: record34, "pick up the red cube" at 2 Hz, time spent request-loop start to VLA call (outbound), VLA call to VLA return (backend) and loop start to results published (end-2-end)

Times (ms)	**min**	**mean**	**p95**	**max**
outbound	6.292	7.035	7.15	23.108
backend	132.743	139.992	141.265	364.266
end-2-end	140.784	148.363	149.377	389.507

Table 5.2: Evaluation of end-to-end (E2E), outbound (Out) response times per record and cumulative translation (Trans.) and cumulative rotation (Rot.)

ID	Freq. (Hz)	Task	E2E mean (ms)	E2E p95 (ms)	Out mean (ms)	Out p95 (ms)	Trans. (m)	Rot. (rad)
01	1	go up	7.323	8.055	5.963	6.631	0.248	0.421
02	1	go down	7.361	8.236	5.983	6.635	0.255	0.426
03	1	yaw right	7.206	8.105	5.882	6.834	0.039	1.248
04	1	yaw left	6.951	7.650	5.699	6.357	0.094	1.191
05	1	pitch up	7.196	8.104	5.913	6.578	0.281	0.836
06	1	pitch down	7.210	8.340	5.939	6.692	0.076	0.238
07	1	roll right	7.313	8.361	5.953	6.725	0.104	0.193
08	1	roll left	7.275	8.143	5.935	6.536	0.008	0.043
09	1	close gripper	7.351	8.323	5.982	6.720	0.081	0.138
10	1	open gripper	7.425	8.220	6.007	6.720	0.029	0.048
11	5	go up	6.065	6.631	4.965	5.504	0.121	0.197
12	5	go down	6.152	6.680	5.096	5.550	0.138	0.232
13	5	yaw right	5.941	7.150	4.935	5.790	0.098	1.003
14	5	yaw left	6.189	6.948	5.122	5.702	0.096	1.251
15	5	pitch up	6.156	6.853	5.141	5.810	0.211	0.755
16	5	pitch down	6.067	6.610	5.091	5.573	0.065	0.285
17	5	roll right	6.055	6.600	4.996	5.479	0.007	0.047
18	5	roll left	6.123	6.730	5.106	5.696	0.001	0.010
19	5	close gripper	6.169	6.828	5.047	5.594	0.001	0.001
20	5	open gripper	6.049	6.773	5.045	5.664	0.000	0.001

Because both the backend and client ran on the same host, this overhead was minimal. When the backend is hosted on a separate machine, additional network round-trip latency would be expected, typically a few tens of milliseconds on a local network and higher in remote deployments. No dropouts, repeated requests, or timeouts occurred in any of the evaluated records within the first 50 requests used for analysis.

Model Inference Latency

When the backend executed a VLA model, the inference stage dominated total processing time. A representative example using the base *OpenVLA-7b* model [Kim+24] at 2 Hz showed an average outbound delay of 7 ms, a backend inference time of approximately 140 ms, and an overall end-to-end loop time of about 150 ms. Table 5.1 lists the detailed timing results for this example, while Table 5.3 summarizes all recorded runs. Across these recordings, outbound times averaged 6–8 ms, backend times 130–150 ms, and the 95th percentile

Table 5.3: VLA evaluation of end-to-end (E2E) and backend processing times for Duplo and Cube tasks.

ID	**Freq. (Hz)**	**Task**	**Cam pos.**	**Fine-tune**	**E2E mean (ms)**	**E2E p95 (ms)**	**Backend mean (ms)**	**Backend p95 (ms)**
21	2	duplo	center	on	149.475	150.135	140.623	141.823
22	1	duplo	center	on	144.257	149.566	135.227	138.901
23	5	duplo	center	on	145.456	148.280	137.483	140.243
24	1	duplo	left	on	150.019	148.937	141.096	140.514
25	2	duplo	left	on	144.262	149.164	135.975	140.359
26	5	duplo	left	on	143.285	144.830	135.436	136.714
27	1	duplo	center	off	147.540	144.421	138.950	136.349
28	2	duplo	center	off	142.934	144.705	134.655	135.992
29	5	duplo	center	off	142.768	144.299	135.067	136.133
30	1	duplo	left	off	143.319	144.955	134.715	136.212
31	2	duplo	left	off	142.601	144.295	134.340	135.922
32	5	duplo	left	off	142.477	143.437	134.903	135.745
33	2	cube	left	off	142.565	143.673	134.305	135.409
34	2	cube	center	off	148.363	149.377	139.992	141.265

never exceeded 160 ms.

No difference was observed between runs with or without local fine-tuning files. Loading additional model weights introduced no measurable delay, and no missed responses or failures were recorded within the evaluated subset.

Frequency and Asynchronous Behavior

Request frequency had no effect on inference time. The backend produced consistent timings at 1, 2, and 5 Hz. Because the node publishes results asynchronously, the bridge continued sending requests while the robot executed previous motions, preventing blocking or accumulation of delay. In recordings with shorter request intervals (200 ms), smaller total translations and rotations were observed, which can be attributed to incomplete physical motion rather than software latency.

5.4 Modularity

All model-specific code resides in the **vla_wrapper.py** of the backend, which encapsulates loading, preprocessing, prompt formatting, and post-processing. To integrate a different VLA, only a new wrapper has to be implemented exposing the same **predict()** signature and returning cartesian and gripper deltas in the same JSON schema. The bridge client and its service interfaces therefore remain unchanged.

Because the backend is a lightweight Flask server, the same architecture can host any number of model wrappers, either switched by name or run on different ports. The backend can also be deployed remotely, for instance on a GPU cluster. This is most simply realized through SSH port-forwarding, allowing the bridge node inside the ROS2 container to communicate securely with a potentially external inference host without modification.

5.5 Summary

The experiments verified that the bridge could successfully connect the ROS2 client to the backend, configure and start the request loop through its service interfaces, and exchange all data required for VLA inference. Image, joint, and prompt information were transmitted correctly, and the returned deltas were applied to the robot. Within the first fifty records evaluated for each run, no communication errors, timeouts, or dropped requests occurred.

While the communication and data handling worked as intended, the control stack showed several inconsistencies. Fixed deltas resulted in positional and rotational drift, and at higher request frequencies, overlapping updates led to further deviation and gripper lag.

For model-based evaluations, both the fine-tuned and base *OpenVLA-7b* variants produced motion but failed to achieve task-directed behavior. The fine-tuned model quickly converged to nearly static outputs, whereas the base model generated continuous yet unstructured movement.

Latency measurements confirmed stable timing across all configurations, with an average communication overhead of 6–8 ms, backend inference around 140 ms, and total loop times below 160 ms.

The modular design simplifies adaptation to other VLA models and would support distributed execution, confirming that the bridge fulfills its goal of separating ROS-specific logic from model-specific implementation.

Together, these results demonstrate that the bridge architecture functioned conceptually but revealed control-level instability and the backend could not produce actionable deltas for our setup, providing the foundation for further discussion in the next chapter.

Chapter 6

Discussion and Outlook

This chapter discusses the findings of the evaluation, which had to narrow its focus from end-to-end model performance to the functionality of the ROS2–VLA bridge itself, as the model could not produce reliable action deltas in our lab. Despite this limitation, the recorded results offer valuable insight into the bridge's communication reliability, control behavior, and timing characteristics. The following sections interpret these outcomes, relate them to the system design, and outline the broader implications for future integration of AI in robotics.

6.1 Interpretation of Results

The evaluation confirms that the ROS2–VLA bridge functions reliably as a communication interface. The exchange between the ROS2 client and the backend remained stable throughout all tests, with no data loss or failed requests. The encoding, transmission, and response handling via the Flask server introduced only minimal delay, validating the design choice of a lightweight HTTP-based interface.

Control behavior, however, revealed several inconsistencies. A mismatch appeared between the end-effector position calculated from joint states and the position actually reached after applying the predicted deltas. Adjusting controller stiffness or tool weight did not remove the error, suggesting that either the configuration of the robot, sensor inaccuracies in the hardware (5 years old), limitations within the Cartesian impedance controller itself or a combination of those could be at fault. Expected timing related issues were visible in gripper actions, where overlapping ROS2 move or stop action calls caused delays or no motion at all. These issues indicate that synchronization between the bridge's request loop and ROS2 control primitives remain a major challenge.

Latency measurements show that backend inference is the dominant performance bottleneck. The inference time of roughly 140 ms per request far exceeds the communication overhead of less than 10 ms. Even when accounting for a local network connection that could add several tens of milliseconds, reasonable request frequencies of up to around 5 Hz remain achievable. This makes the bridge technically viable for low-speed, semi-continuous control but not for high-speed reactive behavior.

On the model side, the fine-tuned VLA repeatedly froze, returning near-zero deltas. This behavior likely originates from an incorrect or inconsistent `unnorm_key` during training or insufficient normalization of the fine-tuning data. The base model, by contrast, produced continuous but unstructured motion, confirming that the bridge can process inference with reasonable frequency, even though the model outputs themselves were not meaningful. The inability to trace these internal model logic illustrates the black-box character of this VLA and the absence of transparent diagnostic tools.

Conceptually, the results highlight a deeper incompatibility between the control paradigms of ROS2 and VLA models. ROS2's infrastructure, as illustrated by the gripper action interface, is oriented toward deterministic sequences of discrete actions. VLA models, in contrast, operate in short iterative loops with small, continuous updates. This mismatch suggests that successful future integration will either require adaptations of existing ROS2 components or new abstractions that can reconcile reactive inference with deterministic control.

6.2 Model Behavior and Generalization

While the preceding section focused on the technical reliability of the bridge itself, this part examines the behavior of the VLA model. Despite stable communication and consistent data flow, neither the base nor the fine-tuned model produced meaningful, goal-directed motion. The fine-tuned VLA repeatedly returned near-zero deltas after only a few iterations, effectively freezing. In contrast, the base *OpenVLA-7b* model generated continuous but unstructured movement that did not correlate with the prompted task.

Although fine-tuning was formally outside the scope of this thesis, a limited experiment was conducted to explore its potential benefits. The training run had to be resumed under time constraints and without full documentation after an initial setup by a colleague, which limits the interpret-ability of the resulting model. Consequently, it remains uncertain whether the observed freezing behavior originated from an incorrect or inconsistent *unnorm_key*, insufficient normalization during training, or a general lack of data diversity.

A degree of inaccuracy was also observed in the Cartesian control stack, particularly in the conversion between joint states and end-effector poses. However, such deviations are not expected to cause complete inference collapse, as VLAs are typically robust to moderate sensor noise and calibration drift. The backend debugging interface confirmed that image data were correctly received. After converting, and resizing according to *OpenVLA* specifications for model input, faulty image processing seems unlikely.

A more plausible explanation lies in a combination of model-level factors: domain shift between the training data and the physical lab environment, differences in camera calibration, lighting, and workspace geometry, and the inherent brittleness of the fine-tuned weights. The results suggest that while the bridge transmitted valid data, the models themselves failed to generalize beyond their original training distribution.

This outcome underlines a key limitation of current VLA architectures: their dependency on well-aligned training domains and the lack of interpretability when behavior diverges from expectations. Without transparent diagnostic tools or accessible intermediate representations, debugging such models remains largely speculative. Consequently, the bridge serves not only as a control interface but also as a diagnostic framework to observe and characterize these generalization failures in real time.

6.3 Architecture

The experimental results support the intended modular design of the ROS2–VLA bridge. The architecture separates robot control from model inference through a simple REST-based interface, allowing each side to evolve independently. During testing, this modularity proved functional: as long as the backend implemented the expected `/predict` endpoint and returned a compatible JSON structure, the client node operated without modification. This confirmed that the interface design is general enough to support future VLA implementations with only minor backend adjustments.

While the communication layer behaved as intended, several observations pointed to control inaccuracies accumulating during continuous motion. These deviations likely stem from the repeated computation of new poses based on slightly outdated or noisy joint readings. An improvement would be to derive each new pose from the last successfully published position rather than exclusively from current sensor states, or to combine both values where appropriate. This could reduce drift and maintain a more consistent relationship between predicted deltas and actual motion.

The gripper control interface presented another point for refinement. Using the ROS2 action mechanism introduced small delays and occasional overlaps when commands arrived faster than they could be executed. Replacing this mechanism with a dedicated topic for publishing target widths would provide a more direct and time-stable interface. Alternatively, queue-based handling for both pose and gripper commands could ensure ordered execution under high request frequencies.

Overall, the architecture validated its main design principle: maintaining a clear separation between deterministic control and learned inference enables flexibility and re-usability. At the same time, the results highlight that improving control precision and synchronization will be key to ensuring reliable behavior as more capable VLA models are integrated through the same interface.

6.4 Theoretical Reflections

The observations made in this work highlight a deeper conceptual divide already introduced in Chapter 2. ROS2 represents a deterministic and model-based view of control: each movement is part of a predefined plan derived from explicit state information, and execution is expected to follow precise timing and verification. VLA models, by contrast, rely on reactive inference. They generate immediate predictions from perception and language context rather than following a precomputed sequence.

During testing, this difference became apparent in how the two systems handled time and intent. The ROS2 control stack expects a precise and ordered chain of actions—pose goals, gripper commands, and controller updates—executed one after another. The VLA backend instead produced continuous micro-adjustments that had no fixed endpoint or guarantee of convergence. This mismatch forced the bridge to interrupt actions or scale predictions so they could be executed within reasonable intervals. The need for such adaptation suggests that even when the technical interface works, the conceptual assumptions of each side remain in conflict.

This conflict is not just specific to this particular implementation. The bridge's difficulties with drift accumulation, synchronization, and timing reflect a broader incompatibility: deterministic controllers rely on exact execution of complete instructions, while learning-based models continuously adapt their output to perception and context. These patterns demonstrate that successful integration will require more than lower latency or improved training—it will need architectural mechanisms that reconcile reactive inference with planned control.

Looking forward, bridging these paradigms may require intermediary layers that mediate between inference and control—hybrid architectures or supervisory schemes that can enforce stability while preserving adaptivity. Whether such mechanisms will emerge as part of future ROS2 packages, or through entirely new frameworks, remains an open question. As learning-driven autonomy continues to expand, it becomes increasingly relevant to ask whether ROS2, built around deterministic scheduling and reproducibility, can remain the primary platform for such systems, or if robotics will need new foundations better aligned with continuous, inference-based control.

6.5 Outlook and Future Work

The most immediate next step is to repeat the evaluation with a correctly fine-tuned or otherwise functional VLA capable of consistent, goal-directed motion. Such a model would allow a complete end-to-end validation of the system and demonstrate the bridge's intended purpose: continuous closed-loop control between perception, language, and action. Further improvements should address synchronization and stability in the control loop. Using the last published pose in combination with current joint readings could reduce drift and timing inconsistencies. The gripper control interface could also be redesigned around a dedicated width topic or a queue-based mechanism to ensure consistent execution under varying request frequencies.

Beyond these refinements, further experiments should focus on broader research and evaluation questions. Better generalization across physical setups would allow systematic comparison between models and integration strategies. Comparative experiments could test multiple VLA models within the same ROS2 bridge framework, separating model-specific failures from architectural ones. Using other packages or the built in sim capabilities of *multipanda_ros2* one could compare simulation to real-world performance. To do so, a cartesian impedance controller would need to be implemented, as this interface is currently not part of the built-in simulation capabilities of the package [mul25]. Evaluating how inference latency, controller timing, and synchronization scale in multi-arm or distributed setups would also help clarify the bridge's performance limits.

Finally, the conceptual tension between deterministic control and reactive inference points toward a longer-term research direction. As VLA models improve, bridging these paradigms may require hybrid control structures that combine learned inference with deterministic safety and planning layers. Whether future ROS2 packages will evolve to support such mechanisms, or whether entirely new frameworks will emerge that better reflect inference-driven autonomy, remains an open question. The current bridge provides a practical foundation for exploring these developments, demonstrating both the feasibility and the friction at the intersection of classical robotics and artificial intelligence.

Bibliography

[Bla+24] Kevin Black et al. π_0 *$AVision - Language - ActionFlowModelforGeneralRobotControl$*. arXiv:2410.24164 [cs]. Nov. 2024. DOI: 10 . 48550 / arXiv . 2410 . 24164. URL: http://arxiv.org/abs/2410.24164 (visited on 10/12/2025).

[Bro+23a] Anthony Brohan et al. *RT-1: Robotics Transformer for Real-World Control at Scale*. arXiv:2212.06817 [cs]. Aug. 2023. DOI: 10.48550/arXiv.2212.06817. URL: http://arxiv.org/abs/2212.06817 (visited on 10/20/2025).

[Bro+23b] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control*. arXiv:2307.15818 [cs]. July 2023. DOI: 10.48550/arXiv.2307.15818. URL: http://arxiv.org/abs/2307.15818 (visited on 09/05/2025).

[Dev+19] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. en. In: *Proceedings of the 2019 Conference of the North*. Minneapolis, Minnesota: Association for Computational Linguistics, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: http://aclweb.org/anthology/N19-1423 (visited on 08/25/2025).

[Fra] Franka Robotics GmbH. *Franka documentation*. en. URL: https://franka.de/documents (visited on 10/20/2025).

[Gat98] Erann Gat. “On Three-Layer Architectures”. en. In: (1998). URL: https://flownet.com/gat/papers/tla.pdf (visited on 10/20/2025).

[Kim+24] Moo Jin Kim et al. *OpenVLA: An Open-Source Vision-Language-Action Model*. arXiv:2406.09246 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2406.09246. URL: http://arxiv.org/abs/2406.09246 (visited on 08/25/2025).

[Liu+23] Haotian Liu et al. “Visual Instruction Tuning”. In: *Advances in Neural Information Processing Systems*. Ed. by A. Oh et al. Vol. 36. Curran Associates, Inc., 2023, pp. 34892–34916. URL: https://proceedings.neurips.cc/paper_files/paper/

2023/file/6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf.

[Mac+22] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[MKW25] Robert Mahony, Jonathan Kelly, and Stephan Weiss. *Galilean Symmetry in Robotics*. arXiv:2510.10468 [cs]. Oct. 2025. DOI: 10.48550/arXiv.2510.10468. URL: http://arxiv.org/abs/2510.10468 (visited on 10/17/2025).

[Mor95] Hans Moravec. *Mind children: the future of robot and human intelligence*. eng. 4. print. Cambridge: Harvard Univ. Press, 1995. ISBN: 978-0-674-57618-6.

[MS22] Matthias Mayr and Julian M. Salt-Ducaju. *A C++ Implementation of a Cartesian Impedance Controller for Robotic Manipulators*. arXiv:2212.11215 [cs]. Dec. 2022. DOI: 10.48550/arXiv.2212.11215. URL: http://arxiv.org/abs/2212.11215 (visited on 10/20/2025).

[mul25] multipanda. *tenfoldpaper/multipanda_ros2*. original-date: 2024-04-05T07:21:09Z. Sept. 2025. URL: https://github.com/tenfoldpaper/multipanda_ros2 (visited on 10/20/2025).

[ONe+25] Abby O'Neill et al. *Open X-Embodiment: Robotic Learning Datasets and RT-X Models*. arXiv:2310.08864 [cs]. May 2025. DOI: 10.48550/arXiv.2310.08864. URL: http://arxiv.org/abs/2310.08864 (visited on 10/20/2025).

[Sha+25] Rui Shao et al. *Large VLM-based Vision-Language-Action Models for Robotic Manipulation: A Survey*. arXiv:2508.13073 [cs]. Sept. 2025. DOI: 10.48550/arXiv.2508.13073. URL: http://arxiv.org/abs/2508.13073 (visited on 10/20/2025).

[SMF21] Mohit Shridhar, Lucas Manuelli, and Dieter Fox. *CLIPort: What and Where Pathways for Robotic Manipulation*. arXiv:2109.12098 [cs]. Sept. 2021. DOI: 10.48550/arXiv.2109.12098. URL: http://arxiv.org/abs/2109.12098 (visited on 09/05/2025).

[Vas+17] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Appendix A

KI-Nutzung

Art der Nutzung	**Beschreibung**
Korrektur Rechtschreibung & Grammatik	Verwendung über ChatGPT.com (Plus subscription): • OpenAI GPT-5 Verwendung der in Overleaf integrierten KI-Assistenz: • Writefull mit GPT-3 Nutzungszeit: September – Oktober 2025
Unterstützung bei der Softwareentwicklung	Verwendung über ChatGPT.com (Plus subscription): • OpenAI GPT-5 • OpenAI GPT-4.0 Nutzungszeit: Juni – Oktober 2025

Table A.1: Übersicht der KI-Nutzung in meiner Arbeit