Universiteit Leiden

Master Computer Science

Edge-VLA: Offline Multimodal Robotic Manipulation with Vision–Language Model on Embedded Hardware

Name:	Chen Chen
Student ID:	s4139399
Date:	16/12/2025
Specialisation:	Artificial Intelligence
1st supervisor:	Dr. E.M. Bakker
2nd supervisor:	Prof. dr. M.S. Lew

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Einsteinweg 55
2333 CC Leiden
The Netherlands

Abstract

Embodied Vision-Language-Action (VLA) systems have shown strong semantic generalization in robotic manipulation, but most rely on cloud resources or high-end GPUs. This thesis investigates whether multimodal reasoning and language-conditioned robot control can run fully offline on resource-constrained edge hardware. We present Edge-VLA, a modular framework that integrates perception, vision-language reasoning, and robot motion planning on an NVIDIA Jetson Orin NX with a locally deployed Qwen 2.5-VL (3B) model.

We evaluate Edge-VLA in Gazebo simulation with a UR5 robot arm on our RT2-Sim24 Task Set, a suite of 24 language-conditioned pick-and-place tasks adapted from RT-2. Three experiments examine feasibility, performance–efficiency trade-offs, and robustness. The Jetson-based configuration achieves an average success rate of 93% on our RT2-Sim24 Task Set while achieving competitive end-to-end latency relative to larger GPU and cloud setups. Robustness tests across layout, lighting, camera pose, and clutter perturbations indicate moderate resilience under layout and lighting changes, but pronounced sensitivity to viewpoint variation and cluttered scenes.

Overall, the results show that fully offline, language-driven robot manipulation is feasible in our simulation study on embedded hardware for structured tabletop tasks, while highlighting challenges in viewpoint and clutter robustness.

Contents

1 Introduction

Advancements in artificial intelligence have revealed the limitations of perception-only or cognition-only models in handling complex, interactive, and autonomous tasks [1]. Embodied AI [2] has emerged as a promising paradigm for developing agents that continuously perceive, reason, and act within their environments. It emphasizes the importance of the integration of perception, reasoning, and control to enable adaptive and context-aware behavior. Representative applications include navigation, object manipulation, assembly, and interactive question answering [3].

Recent progress in large-scale pretrained models has further driven this paradigm through the development of vision-language-action (VLA) systems that combine visual perception, language understanding, and action generation within unified frameworks. Compared with reinforcement-learning or behavior-cloning approaches, VLAs demonstrate stronger generalization and enable flexible, language-based interaction [4]. Systems such as RT-2 [4], OpenVLA [5], and π_0 [6] have shown that natural-language instructions can directly guide robotic control through multimodal reasoning. In particular, RT-2 fine-tunes a large vision-language model PaLI-X / PaLM-E [2], [7] on paired web and robotic data and executes inference directly on real-world robots, achieving semantic generalization across object handling tasks like pick, place, open and close.

However, most existing systems, such as PaLM-E [2], RT-2 [4] and OpenVLA [5], depend on cloud-based resources and high-end hardware, limiting their deployment on platforms with limited computational resources. Moreover, prior research [4], [5], [6] has largely emphasized model architecture and benchmark performance, with relatively little attention to end-to-end implementation in embedded settings. In addition, the development of VLA models typically relies on massive datasets, extensive training and task-specific fine-tuning, which confines progress to organizations with substantial computational resources [3].

These challenges point to an important question: Can we achieve efficient multimodal reasoning and robotic control when computational resources are limited and local only?

To explore this question, in this work, we proposed and implemented a modular, fully local multimodal robotic system that integrates perception, reasoning, and control on an embedded platform. Our study addressed three research questions:

1. **Feasibility.**
   Can a locally deployed multimodal model achieve high success rates on a simulated task set adapted from large-scale embodied systems like RT-2 [4]?
2. **Performance–Efficiency Trade-off.**
   How does task performance and latency vary across model scales and platforms, and what trade-offs emerge between accuracy and computational efficiency?
3. **Robustness.**
   Does the proposed system maintain consistent manipulation performance under changes in scene layout, gripper physical parameters, camera viewpoint, and visual appearance such as illumination variation and background clutter?

This research aims to assess the feasibility, efficiency and robustness of executing multimodal reasoning for selected tasks (Section 5.2.2) on embedded resource-limited hardware. The task set in this work is adapted from RT-2 [4] and is hereafter referred to as the `RT2-Sim24 Task Set`. It consists of 24 language-conditioned pick-and-place tasks in a tabletop scene, where the robot moves a single object relative to printed symbols, numbers, logos, or human faces.

A system is proposed in which perception, language understanding, and motion planning are combined in a modular way. It consists of a framework that integrates three core modules: a perception module for visual recognition and spatial modeling, a reasoning module running a locally deployed vision-language model (Qwen 2.5-VL [8]), and a control module for semantic-conditioned action execution through motion planning based on ROS [9]. All experiments are conducted in a Gazebo [10] simulation environment using our `RT2-Sim24 Task Set`, to ensure that results remain comparable though operating under a simulated, resource-constrained setting. The framework is deployed on an NVIDIA Jetson Orin NX [11] platform.

The main contributions of this thesis are summarized as follows:

1. **Feasibility of offline embodied reasoning.**
   The study shows the feasibility of vision–language reasoning and control when executed entirely on an embedded device, with promising initial results on the practical viability of offline multimodal embodied AI.
2. **Empirical validation on RT2-Sim24 Task Set.**
   Experiments on our `RT2-Sim24 Task Set` executed in a simulator show that the lightweight Qwen 2.5-VL [8] 3B model achieves an average 93% task success rate while operating under computational constraints (executed on Jetson Orin NX 16GB [11]). Robustness tests further show that, relative to this 93% baseline, the success rate remains acceptable under randomized object layouts (85%), perturbed gripper parameters (77%), and lighting changes (96%), but drops markedly to 53% in cluttered scenes and to 45% when the camera pose is tilted by up to ten degrees, indicating pronounced sensitivity to heavy visual clutter and viewpoint changes.
3. **Modular and reproducible architecture.**
   The proposed framework implements perception, reasoning, and control within a unified pipeline, offering a reproducible and extensible architecture for future research on resource-efficient embodied AI and sim-to-real adaptation.

The remainder of this thesis is organized as follows. Chapter 2 reviews related work on multimodal foundation models and their application strategies in embodied AI. Chapter 3 introduces the fundamental concepts that underpin this study, including embodied intelligence, edge AI and prompt engineering. Chapter 4 presents the RT-2 [4] system as a representative VLA baseline and summarizes its model architecture, training process and task setting. Chapter 5 describes the proposed Edge-VLA framework based on Qwen 2.5-VL [8], including the system instantiation, task set, software implementation and hardware deployment. Chapter 6 outlines the experimental framework and evaluation metrics. Chapter 7 reports the results of the simulation experiments. Chapter 8 provides an integrative discussion, covering the major findings, system characteristics, engineering insights, limitations and directions for future research. Chapter 9 concludes the thesis.

2 Related Work

2.1 Evolution of Multimodal Foundation Models

Before discussing specific robotic applications, we first review the evolution of the foundation models that enable current embodied intelligence. Modern embodied systems increasingly follow a progression from LLMs → VLMs → VLAs, moving from language understanding, to vision–language grounding, to action generation [4], [12]. Large Language Models (LLMs) acquire semantic comprehension and reasoning capabilities through massive text pretraining, establishing the foundation for natural language interaction in robotics. Vision-Language Models (VLMs) extend this by integrating visual and linguistic modalities, enabling open-vocabulary environmental perception and scene understanding. Building upon these, Vision-Language-Action models (VLAs) incorporate action generation to form a complete "perception-reasoning-control" pipeline for robotic application.

Our system design closely follows this technological progression. On the embedded (Jetson Orin NX [11]) platform a lightweight VLM model is deployed to process language instructions and environmental understanding, while our VLA framework generates executable action sequences for a specific robot executing a task.

2.1.1 Large Language Models (LLMs)

Advances in large language models (LLMs) like GPT-4 [13] have transformed robotics by enabling unified interfaces for communication, perception, planning, and control [14]. Trained on vast internet-scale datasets, LLMs exhibit emergent abilities and commonsense knowledge, allowing robots to handle complex tasks in open-ended environments beyond rigid pre-programmed instructions [15]. In human-robot communication, modern LLMs such as PaLM [16] and GPT-4 [13] surpass early models like BERT [17]. They support dialogue-based instruction following and clarification, enable natural language interaction, and translate ambiguous commands into executable actions [18]. In addition, by integrating multimodal inputs through vision–language models such as CLIP [19] and AudioCLIP [20], they enhance scene understanding and support cross-modal tasks [21], [22]. LLMs are also shown to be capable of generating emotionally nuanced responses, thereby advancing social robotics [23].

In perception, LLMs enable open-vocabulary object recognition and active perception, allowing robots to reason about objects (e.g., selecting a ceramic vase as a paperweight [24]) and request missing environmental information [25], overcoming limitations of traditional passive systems [26]. For planning and control, LLMs replace rigid symbolic planners like STRIPS [27] by generating flexible action plans from natural language, with frameworks like LLM+P balancing linguistic flexibility and logical rigor [28]. In motion control, LLMs support diverse approaches, from direct trajectory generation [29] to guiding reinforcement learning with subgoal descriptions [30], while safety mechanisms like kinematic constraints ensure reliable outputs.

2.1.2 Vision–Language Models (VLMs)

Large Language Models (LLMs) have revolutionized natural language processing, but real-world 'intelligent' systems require multimodal processing capabilities. This has led to the development of Vision-Language Models (VLMs), which bridge visual and linguistic modalities through various architectural approaches. Early dual-encoder architectures like CLIP [19] used contrastive learning to align images and text in shared semantic spaces, while fusion-encoder architectures such as Flamingo [31] introduced cross-modal attention mechanisms for deeper interaction. The latest generative architectures like PaLM-E [2] directly integrate visual features into LLMs, leveraging their reasoning capabilities for cross-modal tasks, representing an evolution from simple alignment to deep fusion.

VLMs typically employ a two-stage training strategy: pretraining on vast image-text datasets (e.g., LAION-400M [32]) using self-supervised objectives [19], [33] to establish basic cross-modal correspondences, followed by instruction fine-tuning with task-specific data (e.g., LLaVA [34]) to enhance instruction-following abilities. Recent studies have adopted Mixture-of-Experts architectures (e.g., MoE-LLaVA [35]) to balance performance and efficiency. In robotics applications, VLMs enable transformative advances in scene understanding through open-vocabulary recognition [36], [37], human-robot interaction via natural language comprehension (e.g., RT-2 [4]) and interactive troubleshooting [38], and task planning by combining visual input with language reasoning for complex multi-step tasks (e.g., PaLM-E [2]).

In this work, we choose Qwen 2.5-VL [8] as the vision–language model at the core of our multimodal reasoning module. Compared with previously discussed VLMs such as CLIP, Flamingo, and PaLM-E [2], [19], [31], Qwen 2.5-VL provides a unified and efficient architecture for grounded reasoning on resource-constrained hardware. It integrates high-resolution visual encoders with instruction-tuned language decoders, enabling structured outputs such as bounding boxes and spatial coordinates that support robotic manipulation tasks. In our preliminary experiments, we evaluated several open-source VLM candidates on our `RT2-Sim24 Task Set` and found that Qwen 2.5-VL produced more accurate and stable localization results than LLaVA [34], particularly in scenes requiring precise object grounding and compositional reasoning. As shown in Table 1, current benchmark reports further indicate that Qwen2.5-VL-7B-Instruct surpasses GPT-4o-mini [39] on multiple multimodal tasks, and the lightweight Qwen2.5-VL-3B even exceeds the performance of the previous-generation Qwen2-VL 7B model [40], demonstrating strong potential for edge deployment. Combined with its relatively low computational footprint and native support for mixed-precision inference, Qwen 2.5-VL was selected as the fully offline multimodal reasoning component in our proposed Edge-VLA system.

2.1.3 Vision-Language-Action Models (VLAs)

Large language models have evolved from pure text processing to multimodal capabilities, leading to vision-language models (VLMs) for image understanding and visual question answering. This progression has further advanced to vision-language-action models (VLAs), which extend beyond perception to generate actions in physical environments. VLAs represent a step towards human-like embodied intelligence. A typical VLA archi-

		Qwen2.5-VL 7B	Qwen2-VL 7B	GPT-4o Mini	Other Best Open LVLM With Similar Size
College-level Problems	MMMU	58.6	54.1	**60.0**	56.0
	MMMU Pro	**38.3**	30.5	37.6	34.3
Document and Diagrams Reading	DocVQA	**95.7**	94.5	-	93.0
	InfoVQA	**82.6**	76.5	-	77.6
	CC-OCR	**77.8**	61.6	-	61.6
	OCRBenchV2	**56.3**	47.8	43	47.8
General Visual Question Answering	MegaBench	36.8	34.4	**43.1**	34.4
	MMStar	**63.9**	60.7	54.8	62.8
	MMBench1.1	**82.7**	80.7	76.0	79.4
Math	MathVista	**68.2**	58.2	52.5	67.2
	MathVision	**25.1**	16.3	-	19.7
Video Understanding	VideoMME	**65.1**	63.3	64.8	63.3
	MMBench-Video	**1.8**	1.4	-	1.7
	LVBench	**45.3**	-	-	38.4
	CharadesSTA	43.6	-	-	**48.4**
Visual Agent	AITZ	**81.9**	-	-	53.3
	Android Control	**60.1**	-	-	**61.5**
	ScreenSpot	84.7	55.3	-	**89.5**
	ScreenSpot Pro	29.0	1.6	-	**35.7**

		Qwen2.5-VL-3B	InternVL2.5-4B	Qwen2-VL-7B
College-level Problems	MMMU	53.1	52.3	**54.1**
	MMMU Pro	**31.6**	32.7	30.5
Document and Diagrams Reading	DocVQA	93.9	91.6	**94.5**
	InfoVQA	**77.1**	72.1	76.5
	CC-OCR	**74.5**	-	61.6
	OCRBenchV2	**54.3**	-	47.8
General Visual Question Answering	MegaBench	28.9	-	**34.4**
	MMStar	55.8	58.3	**60.7**
	MMBench1.1	77.6	79.3	**80.7**
Math	MathVista	**62.3**	60.5	58.2
	MathVision	**21.2**	20.9	16.3
Video Understanding	VideoMME	61.5	62.3	**63.3**
	MMBench-Video	1.6	**1.7**	1.4
	LVBench	**43.3**	-	-
	CharadesSTA	**38.8**	-	-
Visual Agent	AITZ	**76.9**	-	-
	Android Control	**63.7**	-	-
	ScreenSpot	**55.5**	-	55.3
	ScreenSpot Pro	**23.9**	-	1.6

Table 1: Comprehensive benchmark comparison of Qwen2.5-VL [8] models at both 7B and 3B scales against representative open-source multimodal systems. The upper table compares Qwen2.5-VL-7B with Qwen2-VL-7B [40], GPT-4o-mini [39] and other baselines, and the lower table compares Qwen2.5-VL-3B with InternVL2.5-4B [41]and Qwen2-VL-7B. Both tables report scores on the official Qwen2.5-VL multimodal benchmark suite, which includes college-level problems, document and diagrams reading, general visual question answering, math, etc. [42]. The results highlight the strong performance gains of the Qwen2.5-VL series across diverse tasks and show its suitability for the deployment in our Edge-VLA system.

tecture includes three components: a visual encoder (using models like ViT [43] or CLIP [19]), a language encoder (leveraging LLMs like LLaMA [44] or PaLM [16]), and an action decoder [3]. The key innovation lies in aligning and fusing multimodal representations. Early models like RT-1 [29] used Feature-wise Linear Modulation (FiLM) layers for visual-linguistic interaction, while advanced models like RT-2 [4] employ Transformer-based cross-modal attention mechanisms.

VLAs offer significant advantages over traditional robotic systems. They demonstrate improved generalization, as shown by Gato [45], which can play Atari games, generate captions, and perform robotic tasks with a single model. This versatility comes from world knowledge gained through large-scale pretraining. VLAs enable natural human-machine interaction through language interfaces, allowing users to specify complex tasks like "Put the red cup on the table into the dishwasher." VIMA [46] extends this with multimodal task specifications including videos and gestures. Most importantly, VLAs handle long-horizon tasks through hierarchical architectures. High-level planners like PaLM-E [2] decompose abstract instructions such as "Prepare breakfast" into concrete subtasks like "Take out the bread" and "Pour the milk," significantly improving complex task success rates.

Although the VLA systems discussed above demonstrate strong end-to-end action generation capabilities, our approach adopts a different but related perspective. Instead of training a unified policy that predicts continuous low-level motion commands, we implement a modular VLA framework in which the vision–language model produces structured semantic targets that are subsequently executed through classical motion planning. This design, detailed in Section 5, retains the semantic grounding benefits of VLAs while avoiding the need for large-scale policy training. Compared with fully learned VLAs such as RT-1, RT-2 or Octo [4], [29], [47], our framework is more lightweight and suitable for deployment on embedded hardware, while still ensuring language-conditioned manipulation through the integration of perception, reasoning, and control.

2.2 Application Strategies in Embodied AI

Building on these foundation models, recent researches in robotics often follow two main approaches. The first involves fine-tuning Vision–Language models (VLMs) to specialize them for robotic applications, transforming them into Robotic VLMs or Vision-Language-Action models (VLAs) [2], [4], [5], [6], [29], [47]. These methods adapt model parameters to enable direct perception-decision integration, allowing VLMs to interpret environments and generate actionable robot commands. The second approach utilizes pretrained VLMs in a zero-shot manner, employing prompting techniques for high-level task planning while enforcing physical or logical constraints during execution [48], [49], [50], [51], [52]. Note that in our work we focus on the methods from the second approach. This sub-section discusses the important recent works for these two categories.

2.2.1 Fine-tuned Models for Robotic Perception and Action

RT-1 [29] trains a Transformer that fuses visual observations with language instructions to directly predict discretized robot actions, enabling scalable end-to-end manipulation.

PaLM-E [2] encodes sensor streams and text into a single token sequence processed by an LLM to produce plans that downstream policies execute. RT-2 [4] casts robot actions as text tokens and co-fine-tunes VLMs on vision-language and robotic data to output closed-loop actions with strong semantic generalization. Octo [47] unifies diverse robots and tasks via a Transformer with a diffusion-based action head, supporting varied inputs and action spaces while allowing efficient fine-tuning. OpenVLA [5] treats action prediction as language modeling atop a 7B LLM with fused visual encoders, achieving end-to-end policies trained on large-scale demonstrations. π_0 [6] augments a pretrained VLM with conditional flow matching to generate continuous, high-frequency action chunks for fluent multi-stage manipulation across embodiments.

The works RT-1, PaLM-E, RT-2, Octo, OpenVLA and π_0 share a common goal. They learn an end-to-end policy that maps visual observations and language instructions to robot actions by fine-tuning model parameters on large demonstration datasets. However, these works differ in action representation, with RT-1, RT-2, and OpenVLA using discrete tokens, while Octo and π_0 use continuous trajectories via diffusion or flow matching. They also vary in modality fusion, such as PaLM-E's using interleaved multimodal sentences versus OpenVLA's using VLM-to-LLM adapters. All aim for closed-loop control with minimal reliance on external planners, targeting embodiment generalization, data-driven semantics, and real-time execution. All these listed approaches focus on policy learning through fine-tuning, treating action generation as an internal model capability rather than a downstream optimization.

In this thesis, we use a similar family of selected language-conditioned manipulation tasks as defined and used by RT-2 [4]. RT-2 serves as the most comparable baseline for evaluation under resource-constrained conditions. Contrary to RT-2, our proposed method does not use fine-tuning, and will be evaluated in a simulator.

2.2.2 Task Planning with Pretrained Foundation Models

SayCan [48] composes long-horizon behaviors by multiplying LLM-based task relevance with value-based feasibility, selecting skills that are both meaningful and executable, such as "find a sponge," "pick up the sponge," and "bring it to the user" in response to instructions like "I spilled my drink, can you help?" VoxPoser [49] synthesizes 3D voxel value maps from language and perception via LLM/VLM tools, then uses classical motion planning to produce 6-DoF trajectories. VILA [50] shows that unfreezing the LLM and pretraining on interleaved image–text data with a small amount of text-only instruction improves multimodal alignment and generalization. CoPA [51] leverages VLM commonsense to pick task-oriented grasps and infer spatial constraints for 6-DoF motion optimization, enabling generalizable manipulation. Task-oriented grasping focuses on selecting object parts that are functionally appropriate for the task. For example, CoPA selects the handle of a hammer for striking or the stem of a flower for insertion into a vase. ReKep [52] represents robotic manipulation tasks as sequences of differentiable constraints over semantic 3D keypoints. These tasks include actions like pouring tea, folding clothes, and placing books, which involve multiple spatially and temporally dependent stages. Each stage is defined by sub-goal and path constraints, and the system optimizes a sequence of 6-DoF end-effector poses to satisfy them. Large vision models

(LVMs) detect fine-grained keypoints from RGB-D input, and vision-language models (VLMs) generate constraint functions from language and visual prompts.

The works SayCan, VoxPoser, VILA, CoPA, ReKep adopt a modular approach. They all use pretrained foundation models for high-level semantic reasoning, paired with explicit constraints, value functions, grasp filters, keypoint objectives, or classical motion planning to ensure feasibility. These works differ mainly in their intermediate representations, which refer to the task-level abstractions that connect semantic reasoning to low-level motion control. For example, SayCan uses task feasibility scores over predefined skills, VoxPoser employs LLM/VLM-generated voxel objectives, VILA enhances multimodal reasoning through pretraining, CoPA applies VLM-guided grasp and constraint inference, and ReKep uses differentiable three-dimensional keypoint constraints. All these listed methods separate semantic understanding from low-level execution, following a plan-then-execute pipeline in which pretrained models provide task structure and commonsense, while physical feasibility and safety are ensured by external planners or optimizers. Such a modular structure enables strong zero- or few-shot generalization without end-to-end policy fine-tuning.

The proposed system in this thesis also follows this modular paradigm. Similar to SayCan and VoxPoser, it separates semantic reasoning from motion execution by combining a pretrained vision–language model with classical motion planning through MoveIt [53]. However, the task domains in prior modular studies typically involve skill composition or long-horizon household activities, such as object sorting and tool use, which emphasize symbolic task sequencing rather than precise manipulation. In contrast, this work focuses on short-horizon, language-conditioned pick-and-place manipulation tasks such as the tasks used in RT-2 [4], which offer a more standardized and fine-grained benchmark for evaluating the performance of our proposed system. Adopting the RT-2 task framework thus keeps our simulated task definitions (`RT2-Sim24 Task Set`) aligned with a well established embodied model and ensures reproducibility within a controlled simulation environment.

3 Fundamentals

This chapter establishes the theoretical foundations and core methodologies supporting the proposed offline robotic system. We first introduce Embodied AI, which provides the overarching paradigm for integrating perception, reasoning, and control within a physical agent. Next, we discuss Edge AI, outlining the operational constraints and requirements for deploying intelligent systems locally on embedded hardware. Finally, we present Prompt Engineering, the key technique employed to direct pretrained vision-language models toward structured robotic tasks without parameter fine-tuning.

3.1 Embodied AI

Embodied Artificial Intelligence (Embodied AI) targets intelligent behavior that couples cognition with physical embodiment. Its roots trace to the idea of an "embodied Turing test," which emphasizes competence beyond abstract symbol manipulation toward robust performance in the physical world [54]. Unlike non-embodied systems, embodied agents must perceive, interpret, and act under uncertainty and continuous change, integrating multimodal sensing (vision, audition, touch) with planning and control.

A central requirement for Embodied AI is language–perception–action grounding: mapping linguistic intents to operational targets in 3D space with awareness of object attributes, scene layout, and human priors. This enables dialogue-driven execution and instruction decomposition or planning [55]. Advances in multimodal large models (MLMs) have accelerated this trajectory: VLMs (e.g., CLIP, BLIP-2) provide rich image–text semantics [19], [56], while LLMs improve instruction understanding and reasoning. World models further contribute predictive environment representations, supporting look-ahead action evaluation and capturing physical/causal/temporal regularities [57].

Our proposed framework follows the embodied AI paradigm by implementing a closed-loop system on an embedded platform that fuses visual perception, language understanding, and action execution fully offline. The robot continuously perceives its workspace through an RGB-D camera, interprets natural-language instructions using a locally deployed vision–language model, and converts high-level semantic understanding into low-level actions through motion planning. This process allows the system to transform linguistic intentions such as "move the coffee box to the apple" into grounded spatial goals represented by 3D coordinates. These goals are then executed through coordinated trajectories generated by MoveIt [53] within the ROS [9] framework. Unlike traditional pipelines that separate perception and decision-making, our system maintains active interaction between sensory input and control output. The vision–language model continuously conditions its reasoning on real-time visual data, enabling adaptive and context-aware manipulation. Through this integration, the robot exhibits the key characteristics of Embodied AI: perception grounded in physical space, reasoning constrained by physical feasibility, and actions that alter the environment in meaningful ways.

3.2 Edge AI

While Embodied AI provides the conceptual foundation for integrating perception, reasoning, and action, its deployment on physical agents requires real-time, efficient, and fully autonomous computation. Edge AI addresses this need by executing the required AI processing directly on embedded devices. Rather than depending on remote servers or stable network connections, Edge AI systems execute all perception and control locally, thereby focusing on low-latency response while ensuring robust operation in offline environments. Edge AI is especially suitable or even required for embodied systems deployed in dynamic, resource-limited, or infrastructure-poor settings.

Edge AI performs on-device inference to meet real-time, autonomy, and privacy requirements. While cloud is very suitable for data aggregation and large-scale training tasks such as model pretraining, latency-sensitive control tasks such as obstacle avoidance or grasp execution benefit from local execution, Hybrid cloud–edge designs distribute workloads accordingly, for example by performing high-level policy learning or model updates in the cloud while delegating real-time perception and motion control to the edge device [58]. Compared with cloud-dependent systems, edge deployments sustain operation in bandwidth-limited or disconnected settings and reduce data exfiltration risks, which is crucial for embodied agents operating in dynamic environments [59].

In our work, the robot platform is specially developed to be deployed in an Edge AI setting. All perception, reasoning, and control runs locally on an embedded device, where a local VLM enables semantic instruction following for autonomously executing selected tasks without any network access.

3.3 Prompt Engineering

Prompt engineering has established itself as a fundamental methodology for maximizing the effectiveness of pre-trained large language models (LLMs) and vision-language models (VLMs). This technique focuses on the systematic design of context-specific input instructions (prompts) to precisely guide model behavior while maintaining the original model parameters.

Prompt engineering provides a practical way to adapt model behavior without changing model parameters. Unlike conventional methods that require computationally expensive retraining or extensive fine-tuning, prompt engineering achieves task-specific optimization through carefully structured textual cues. This approach enables AI systems to dynamically adjust their outputs for diverse applications while preserving their core architecture. It enables developers to direct model focus through contextual framing, elicit specialized knowledge without updating the model weight, significantly reduce computational overhead[60].

Effective prompt engineering follows a structured approach consisting of:

1. First, practitioners need to clearly identify the objective by determining what specific output or action they want the model to produce.

2. Next, it is essential to thoroughly evaluate the model's capabilities and limitations to understand what it can realistically achieve.
3. Based on this understanding, an appropriate prompt format should be selected, which could range from brief instructions to more detailed multi-sentence inputs. Including relevant contextual information such as topic details, scenario descriptions, or character backgrounds is also crucial for guiding the model's reasoning process.
4. Finally, the prompt should be tested and refined through multiple iterations to optimize performance.

This methodology can be successfully applied to various AI models including both text-based language models and image generation systems to effectively direct their outputs toward desired results [61].

In our system, prompt engineering plays a central role in connecting semantic reasoning with downstream robotic control. Rather than relying on open-ended natural language queries, we construct task-specific prompts that constrain the model to produce structured outputs compatible with the control pipeline. As described in Section 5.1.1, the prompts follow a fixed template that encodes image resolution, pixel coordinate conventions, and the required JSON schema, ensuring that Qwen 2.5-VL [8] generates deterministic predictions for each manipulation task. This design aligns the model's reasoning process with the perception and planning modules, enabling reliable extraction of object locations and targets for the VLA framework. By combining prompt design with strict output formatting, our system ensures predictable and robust multimodal reasoning suitable for fully offline execution on embedded hardware.

4 VLA Baseline: RT-2

This work adopted RT-2 [4], a VLA system developed by Google DeepMind, as the VLA baseline system. RT-2 is a pioneering system that first introduced the vision-language-action (VLA) model paradigm to the robotics community. As the foundational work that established VLA models, RT-2 demonstrates how these models can effectively transfer knowledge from broad, web-scale datasets to robotic control tasks [4]. RT-2 is a fine tuned real world robotic VLA system that has been evaluated on more than six thousand real robot trials, including standard table top manipulation tasks, generalization to novel objects and instructions, and emergent capability tasks. It provides a strong baseline for evaluating multimodal generalization in robotic manipulation. In our work we used a task set adapted from the task set of RT-2. This chapter outlines the RT-2 system architecture, its training and inference pipeline, and the original task suites used in the RT-2 paper. We also highlight the emergent capability tasks for symbol understanding, reasoning, and person recognition that we adapt as the `RT2-Sim24 Task Set` in our simulated experiments (see Sections 4.4 and 5.2.2).

4.1 RT-2 System Architecture

The core concept of RT-2 is to frame robotic control as a unified vision-language problem. Instead of designing a new architecture, the system is built around a central vision-language model (VLM), as illustrated in Figure 1.

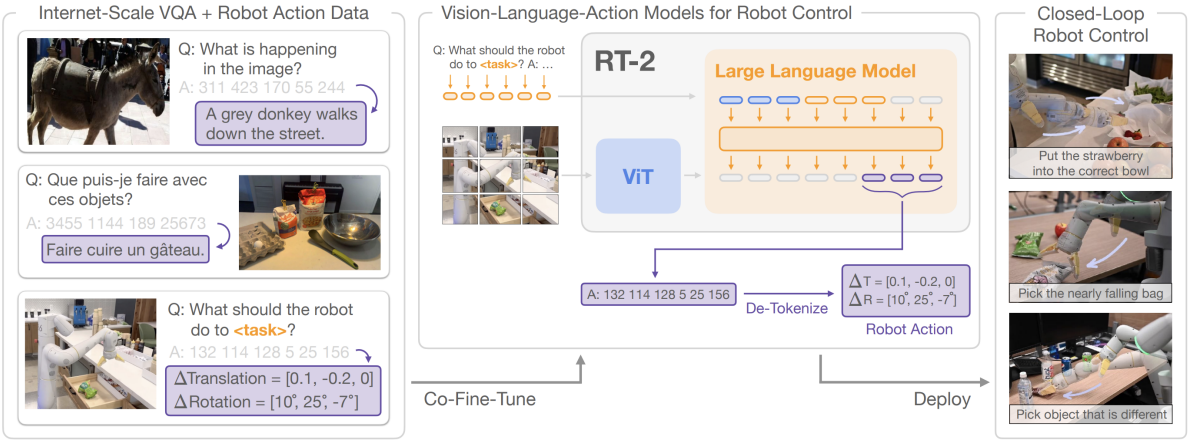


Figure 1: RT-2 overview: the framework represents robot actions as text tokens, enabling joint training with vision-language datasets. It leverages VLM backbones and pretraining to transfer generalization, semantic understanding, and reasoning capabilities to robotic control [4].

The core components of RT-2 are:

- **Vision-Language Model (VLM):** The central part of the system. It consists of two main parts: a Vision Transformer (ViT) to process visual (image) input, and a Large Language Model (LLM) that fuses visual features with text prompts to reason and generate output.

- **Training Dataset:** A mixed dataset referred to as "Internet-Scale VQA + Robot Action Data." This includes both web-scale vision-language tasks like vision-question answering (VQA) and specific robotic trajectory data.
- **Action Representation:** A key conceptual component. Physical actions (e.g., end-effector movements) are discretized and converted into text tokens. This allows actions to be treated as part of the VLM's vocabulary.
- **De-Tokenize Module:** The component that acts as a translator. It receives the text-based action token string generated by the VLM and converts it back into a numerical, executable "Robot Action" command.
- **Robotic Hardware:** The physical robot that executes the final command (a 7-DoF mobile manipulator in the RT-2 study).

4.2 RT-2 Model Training and Inference

The data flow in RT-2's architecture is divided into training and deployment phases:

- **Training Flow:** The VLM is "Co-Fine-Tuned" on the mixed input dataset. This process teaches the model to output either a natural language answer (in response to a VQA task) or a string of action tokens (in response to a robotic task), based on the image-text prompt.
- **Deployment Flow:** During operation, the robot's camera provides an image, which is paired with a text instruction (e.g., "move apple to cup with same color"). Both are fed into the VLM. The model generates an output. If it is a robotic task, the output is a string of action tokens. This string is immediately sent to the "De-Tokenize" module, which converts it into an executable robot action. The action is then executed by the robot, completing the "Closed-Loop Robot Control" cycle.

The RT-2 models are fine-tuned, not trained from scratch. The training process involves a mixture of two data sources:

- **Web-scale VQA Data:** The original web-scale data used to pretrain PaLI-X and PaLM-E [2], [7], including visual question answering, captioning, and other vision-language tasks.
- **Robot Action Data:** A large dataset of robot demonstrations from the earlier RT-1 [29] work. This data was collected by 13 robots over 17 months in an office kitchen environment and annotated with natural language instructions.

For the inference stage, the large model sizes (up to 55 billion parameters) required specific hardware. The models are deployed in a multi-TPU cloud service and queried over a network. RT-2's dependency on large-scale cloud computing is also a key difference compared to our proposed framework deployed offline on embedded device.

4.3 RT-2 Model Instantiation and Hardware

The RT-2 system is not based on a single model but rather a method applied to different VLMs. The experiments this thesis compares against used two specific model instantiations:

- **RT-2-PaLI-X:** Built from PaLI-X [7], using 5-billion and 55-billion parameter versions.
- **RT-2-PaLM-E:** Built from PaLM-E [2], using a 12-billion parameter version.

The robotic hardware used for data collection and evaluation in the RT-2 system was a 7-DoF mobile manipulator. The action space was discretized, representing 6-DoF positional and rotational displacement of the end-effector, as well as the gripper extension state.

4.4 RT-2 Task Set

RT-2 [4] is trained and evaluated on a large suite of manipulation tasks that extend the RT-1 [29] benchmark. It uses the full RT-1 kitchen task set with more than 200 instruction-level tasks such as picking objects, moving them, placing them upright, opening and closing drawers, and placing objects into receptacles. Generalization is then tested under controlled distribution shifts along three axes: unseen objects, unseen visual backgrounds, and unseen environments like a kitchen sink or an office desk. For each axis there are "easy" and "hard" settings, and the corresponding natural-language instructions, which mostly involve variants of pick, move, push, drop and point actions on everyday items. All instructions in this generalization task set are listed in Table 9 in Appendix.

Task Group	Tasks
Symbol Understanding: Symbol 1	move coke can near X, move coke can near 3, move coke can near Y
Symbol Understanding: Symbol 2	move apple to tree, move apple to duck, move apple to apple, move apple to matching card
Symbol Understanding: Symbol 3	put coke can close to dog, push coke can on top of heart, place coke can above star
Reasoning: Math	move banana to 2, move banna near the sum of two plus one, move banana near the answer of three times two, move banana near the smallest number
Reasoning: Logos	move cup to google, move cup to android, move cup to youtube, move cup to a search engine, move cup to a phone
Reasoning: Nutrition	get me a healthy snack, pick a healthy drink, pick up a sweet drink, move the healthy snack to the healthy drink, pick up a salty snack
Reasoning: Color and Multilingual	move apple to cup with same color, move apple to cup with different color, move green chips to matching color cup, move apple to vaso verde, Bewegen Sie den Apfel in die rote Tasse, move green chips to vaso rojo, mueve la manzana al vaso verde, déplacer les frites verts dans la tasse rouge
Person Recognition: Celebrities	move coke can to taylor swift, move coke can to tom cruise, move coke can to snoop dog
Person Recognition: CelebA	move coke can to person with glasses, move coke can to the man with white hair, move coke can to the brunette lady

Table 2: Natural-language instruction sets used in RT-2 [4] to evaluate emergent capabilities, grouped into symbol understanding, reasoning (math, logos, nutrition, color, multilingual), and person recognition tasks.

To study emergent capabilities that go beyond the robot demonstration data, RT-2 is further evaluated on a dedicated instruction set summarized in Table 2. These instructions are grouped into three broad categories. Symbol understanding tasks ask the robot to move objects with respect to abstract symbols or pictures. Reasoning tasks cover several types of semantic reasoning, including simple math, logo-based commands, choices about healthy or sweet food and drinks, and color or multilingual instructions. Person recognition tasks require moving an object to images of specific celebrities or to people with certain visual attributes. These emergent evaluations measure how well RT-2 can transfer web-scale vision–language knowledge into new, semantically rich robot behaviors.

4.5 Comparability: Real-world RT-2 vs Simulated Edge-VLA

In the remainder of this thesis, the RT-2 task suites serve as the semantic reference for our evaluation. However, the reported results in Chapter 7 are not a head-to-head benchmark, because the evaluation settings and task suites are different:

- **Different settings (real vs sim).** RT-2 performs tasks on a real mobile manipulator in kitchen environments. Our experiments are conducted in Gazebo [10] with a UR5 [62] robot arm. Our simulation therefore does not reproduce the full physical and environmental complexity of RT-2.
- **Different task suites.** RT-2 includes the full RT-1 kitchen task set and additional generalization tests as described in Section 4.4. In this thesis, we do not reproduce that full suite. Instead, a subset of the emergent capability instructions in Table 2 is instantiated as a simulated task set called the `RT2-Sim24 Task Set` (Section 5.2.2). Each task in our `RT2-Sim24 Task Set` remains a short, language-conditioned pick-and-place instruction with the same wording and visual targets as in RT-2, but it is executed in our simulation setup.

Our simulation preserves the task intent, visual semantics, and the binary success criterion as in RT-2. However, considering the differences mentioned above, when Chapter 7 shows RT-2 numbers next to our results, RT-2 is used only to indicate the type of semantic capabilities targeted by the instructions, not as a directly comparable performance baseline.

5 Edge-VLA System using VLM Qwen

This chapter presents the novel proposed Edge-VLA system that will be evaluated on our `RT2-Sim24 Task Set` in simulation. Building on the baseline and task set description in the Chapter 4, we detail how a fully offline multimodal control pipeline is designed, implemented and instantiated, and how it supports symbol understanding, reasoning, and person-recognition tasks.

The Edge-VLA system is built using a general six-layer framework that conceptually organizes perception, multimodal reasoning, and control. We then describe the specific instantiation of the framework used in this work, consisting of a camera-only perception setup, the UR5-based manipulation environment [62], and the vision–language model Qwen 2.5-VL [8]. Next, we outline the software and communication architecture, followed by task-specific routines that enable the system to perform designated task in our `RT2-Sim24 Task Set`. Finally, we summarize the hardware deployment.

5.1 General Framework

In this section we introduce our Edge-VLA Framework using an existing VLM and the required prompt engineering for performing selected tasks in our `RT2-Sim24 Task Set`. The proposed Edge-VLA Framework is depicted in Figure 2.

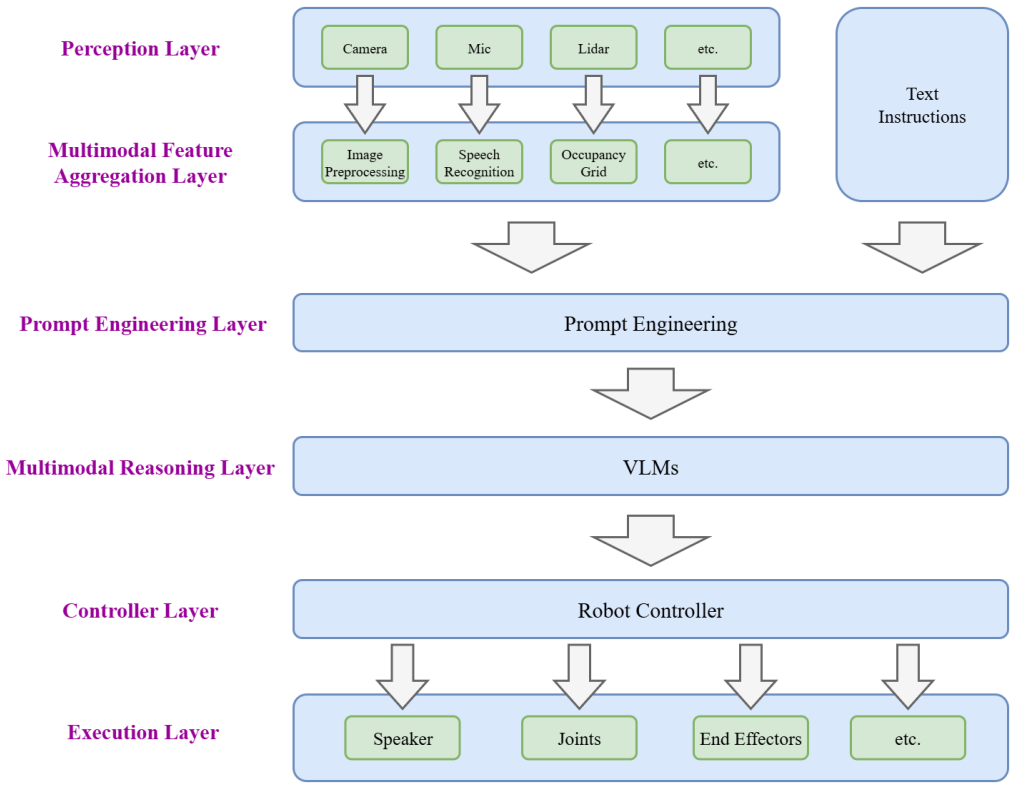


Figure 2: Overall architecture of the proposed Edge-VLA Framework to implement multimodal robot control system, note that in our research the perception layer consists of a camera, and the robotic embodiment is a robotic arm.

The framework supports semantic-driven manipulation. Specifically, grasp-and-place operations guided by natural language instructions and perceptual inputs, such as the selected tasks in our `RT2-Sim24 Task Set` described in Section 5.2.2.

The framework is designed in a modular and interoperable way, such that the VLM, robot control framework, and simulation environment work as independent but compatible modules. This design supports the selection and evaluation of different VLMs, motion planners, etc., in both simulated and real-world deployment. Specifically, it allows the selection of suitable VLMs that support fully offline, multimodal robot control on the designated Jetson platform.

Algorithm 1 Multimodal Local Control Framework Operation

procedure ExecuteTask
Input: Raw Sensory Data ($\mathcal{S}_{raw}$), User Text Instruction ($\mathcal{I}_{text}$)
Output: Physical Actions ($\mathcal{A}_{phys}$)

Layer 1: Perception Layer
$\mathcal{S}_{collected} \leftarrow$ CollectRawData($\mathcal{S}_{raw}$) ▷ From Cameras, Microphones, LiDAR, etc.

Layer 2: Multimodal Feature Aggregation Layer
$\mathcal{F}_{struct} \leftarrow$ ExtractFeatures($\mathcal{S}_{collected}$) ▷ Structured features per modality
$\mathcal{I}_{proc} \leftarrow$ ProcessInstruction($\mathcal{I}_{text}$) ▷ Parallel processing of text

Layer 3: Prompt Engineering Layer
$\mathcal{P}_{unified} \leftarrow$ ConstructPrompt($\mathcal{F}_{struct}, \mathcal{I}_{proc}$) ▷ Generate unified VLM input

Layer 4: Multimodal Reasoning Layer
$\mathcal{L}_{high} \leftarrow$ VLMInference($\mathcal{P}_{unified}$) ▷ Generates High-level Semantic Action Plan

Layer 5: Controller Layer
$\mathcal{C}_{low} \leftarrow$ TranslatePlan($\mathcal{L}_{high}$) ▷ Convert to Low-level Control Commands

Layer 6: Execution Layer
$\mathcal{A}_{phys} \leftarrow$ ExecuteCommands($\mathcal{C}_{low}$) ▷ Via Speakers, Joints, End Effectors

return $\mathcal{A}_{phys}$
end procedure

The framework enables a closed-loop process as given in Algorithm 1 that spans perception, prompt engineering, language and sensory data understanding, leading to actionable instructions for task execution. The architecture consists of six layers: Perception Layer, Multimodal Feature Aggregation Layer, Prompt Engineering Layer, Multimodal Reasoning Layer, Controller Layer, and Execution Layer. As described in Algorithm 1, data from raw sensory inputs is used for multimodal feature extraction and prompt construction. The resulting prompt is processed by the vision-language model for semantic reasoning

and translated into motor robot commands, and executed by the actuators to perform the given task.

5.1.1 Prompt Design for Structured VLM Outputs

Following the four-step prompt engineering workflow introduced in Section 3.3, we formalize our prompt design as a reproducible method that targets reliable, machine-readable VLM outputs for downstream robot control.

1. **Objective definition.**
    - Convert open-ended instruction into a structured visual grounding problem.
    - Enforce a deterministic interface for the controller: the model must output JSON-only with fixed keys because we want the downstream nodes can easily parse them.
    - Define coordinate conventions explicitly (image resolution and pixel coordinate system) so that the coordinates output by VLM are directly usable by task planner.
2. **Model capability & limitation analysis.**
    - The following errors observed during prompt testing: (i) extra explanatory text, (ii) Markdown breaking JSON parsing, and (iii) wrong coordinates output.
    - Embedded deployment constraints motivate concise prompts: longer prompts increase token processing and may raise latency and memory usage on-device.
    - Therefore, the prompt must make the expected output and the required context as explicit as possible by specifying a strict output contract and including only minimal, task relevant information.
3. **Prompt format selection & context encoding.**
    - **Context block:** summarize only the necessary scene assumptions (top-down tabletop, one manipulable object, target region).
    - **Coordinate contract:** state image resolution and pixel-axis conventions to eliminate ambiguity.
    - **Schema block:** specify the exact JSON schema, value types, and a "OUTPUT: Single line JSON only" instruction.
    - **Example output:** provide a minimal JSON example to stabilize formatting while keeping the prompt short.
4. **Iterative testing.**
    - **Prompt screening:** following the three steps above, we designed five prompt variants and evaluated them under the same experimental protocol using end to end task success rate as the selection criterion, and we retained the best performing prompt for all subsequent experiments.
    - **Are these prompts optimal?** We do not claim optimality; instead, we prioritize availability and controller-compatibility in our proposed system.

5.2 System Instantiation and Task Set

5.2.1 System Instantiation

The previous section introduced the generic six-layer framework. Here we describe how it is instantiated in the remainder of this work. Figure 3 summarizes the complete perception–reasoning–control pipeline using a simulated tabletop environment. The system consists of a Kinect RGB-D camera facing top-down, a UR5 robotic arm [62] simulated in Gazebo [10], the Qwen 2.5-VL [8] model for multimodal reasoning, and a ROS/MoveIt-based [9], [53] controller that plans and executes motions for the robot arm.

In the Perception Layer, the system uses a simulated Microsoft Kinect RGB-D camera mounted above the tabletop workspace. The camera provides RGB-D images of size 640×480 pixels, with the origin at the top-left corner (x-axis increasing from left to right, y-axis from top to bottom). The RGB stream is forwarded to the vision–language model for semantic scene understanding, while the depth channel is consumed directly by the task planner in the controller layer to get 3D target locations in the robot frame. The camera intrinsics and the mounting pose are known and fixed, which enables projecting model predictions by the VLM from pixel space into the robot's workspace.

The Multimodal Feature Aggregation Layer is implemented as an image preprocessing stage. It subscribes to the RGB stream in Gazebo [10], captures the latest frame, optionally applies resizing or normalization, and packages the processed image. At this stage, the raw visual observation is converted into a standardized input package that can be consumed by the vision-language model, including the encoded RGB image, its resolution and pixel coordinate convention.

The Prompt Engineering Layer converts the current RGB image and the natural language instruction into a structured prompt for Qwen 2.5-VL [8]. Following the prompt design method in Section 5.1.1, we use a fixed template that provides only minimal task relevant scene context, and enforces a strict output contract. Figure 3 gives an example of prompt used in extended evaluations in **Experiment 1**, the prompt explicitly specifies the image resolution, coordinate system, camera parameters, and qualitative description of the camera pose. Note that for different experimental settings we use different prompts.

The Multimodal Reasoning Layer is realized by the Qwen 2.5-VL [8]model. Given the constructed prompt and the current RGB frame, Qwen performs joint visual and linguistic reasoning: it interprets the tabletop scene, resolves symbolic or semantic references in the instruction (such as "X", "tree", "YouTube logo", or "Taylor Swift"), and predicts the pixel locations of the manipulated object and the selected target. The model outputs the prediction strictly in the JSON schema defined in prompts, which makes the downstream controller agnostic to the internal structure of the language model.

The Controller Layer is implemented using ROS [9]together with MoveIt [53]. It parses the JSON returned by Qwen [8], extracts `object_px` and `target_px`, which denote the predicted center pixel coordinates of the manipulated object and the target location in the current RGB image, respectively, and projects these pixel coordinates into the 3D workspace by using the depth values provided by the RGB-D camera, together with the

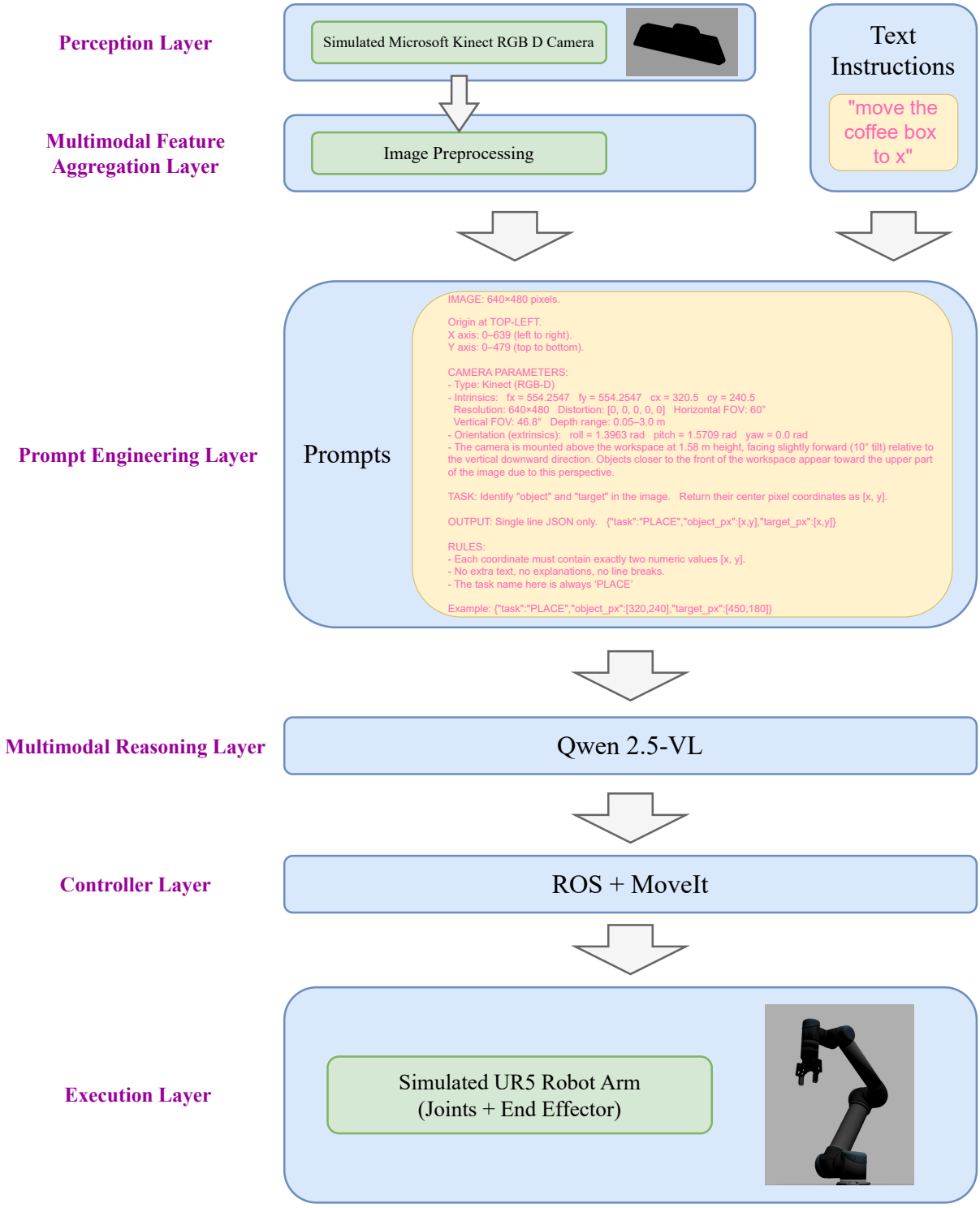


Figure 3: System instantiation of the Proposed Edge-VLA Framework

known camera intrinsics and extrinsics. From these 3D points, the controller synthesizes grasp and place poses for the UR5 end-effector, and formulates motion-planning queries for MoveIt [53] that respect joint limits and collision constraints.

Finally, the Execution Layer comprises the MoveIt [53] motion planner and the simulated UR5 [62] robot arm with its gripper in Gazebo [10]. Based on the target poses from the controller, MoveIt generates collision-free joint trajectories for pick and place. These trajectories are sent to the UR5 joint controllers, which execute the planned motions and move object from its current position to target position specified by the instruction. This closes the loop from language and vision to concrete robot actions within the instantiated six-layer framework.

5.2.2 RT2-Sim24 Task Set

The RT-2 [4] baseline VLA system was originally evaluated on two types of task sets. The first type covers in-distribution and generalization tasks derived from RT-1 [29], along several distribution shifts over objects, backgrounds, and environments. The second type is a dedicated set of "emergent capability" instructions, which groups tasks into three broad semantic categories: symbol understanding, reasoning, and person recognition. All these tasks have been described in Section 4.4.

In this work we focus on the second type of tasks and do not use the first type of the RT-1 [29] task set, as the RT-1 based tasks and their distribution shifts depend on a rich real-world environment with drawers, cupboards, appliances, and diverse objects, which are difficult to reproduce faithfully in our simulated setup with a single UR5 [62] robot arm and a fixed tabletop scene. Instead we focus on evaluating our fully offline, embedded VLM-driven system's performance on semantic understanding and language-conditioned control. The emergent capability tasks isolate these semantic aspects, while keeping the low-level control pattern simple and constant.

Concretely, the system given in Figure 3 implements a subset of the emergent capability instructions from Table 2 as language-conditioned pick-and-place scenarios in simulation. As illustrated in Table 3, We define a fixed set of 24 instructions that are organized into the same three categories as RT-2 [4]. The `Symbol Understanding` category comprises 9 tasks where the robot must move a coffee box relative to printed abstract symbols on the table, such as placing the box on an "X" marker, on a heart, or next to a tree or animal icon. The `Reasoning` category contains 9 tasks that combine simple math and logo-based semantics, for example moving the box to the number that matches "two plus one" or to the printed logo that represents a specific brand or service. The `Person Recognition` category contains 6 tasks in which the system must move the box to the image of a particular celebrity or to a face that matches a visual attribute such as "wearing glasses" or "white hair". This task set is named as `RT2-Sim24 Task Set`.

In our evaluation experiments we use the `RT2-Sim24 Task Set` as defined in Table 3. Note that this is by design very similar to the original task-set listed in Table 2 in Section 4.4 which was used to evaluate RT-2 in [4].

Task Group	Tasks
Symbol Understanding: Symbol 1	move coffee box to X, move coffee box to 3, move coffee box to Y
Symbol Understanding: Symbol 2	move coffee box to tree, move coffee box to duck, move coffee box to apple
Symbol Understanding: Symbol 3	put coffee box close to dog, put coffee box on top of heart, place coffee box above star
Reasoning: Math	move coffee box to 2, move coffee box to the sum of two plus one, move coffee box to the answer of three times two, move coffee box to the smallest number
Reasoning: Logos	move coffee box to google, move coffee box to android, move coffee box to youtube, move coffee box to a search engine, move coffee box to a phone
Person Recognition: Celebrities	move coffee box to taylor swift, move coffee box to tom cruise, move coffee box to snoop dog
Person Recognition: CelebA [63]	move coffee box to person with glasses, move coffee box to the man with white hair, move coffee box to the brunette lady

Table 3: **RT2-Sim24 Task Set:** Natural language instructions used in this work

5.3 Software Implementation

The instantiated system in Figure 3 is realized by a set of modular software components. These components communicate using ROS [9] messages and HTTP requests. Figure 4 summarizes the overall software and communication architecture. On the left, a Gazebo [10] simulation provides the UR5 [62] arm, the tabletop scene and a simulated Kinect RGB-D camera. In the centre, an Ubuntu virtual machine runs the ROS and MoveIt [53] control stack. On the right, a Jetson Orin NX [11] runs the Qwen 2.5-VL [8] inference server under NVIDIA JetPack [64]. Together they form a closed loop from language and vision to robot motion.

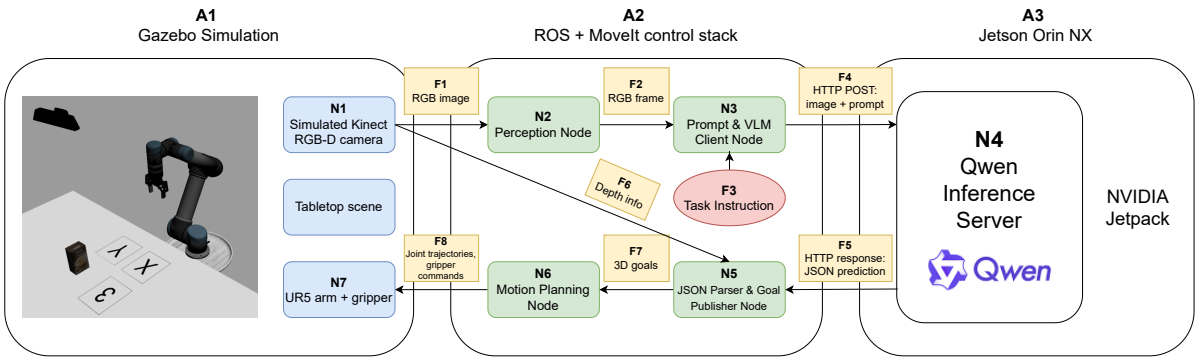


Figure 4: Software and communication architecture of the used Edge-VLA system.

To make the software architecture in Figure 4 more concrete, we walk through one com-

plete trial of a designated task in our `RT2-Sim24 Task Set`. As a running example, we use the symbol understanding instruction `"move the coffee box to X"` in a tabletop scene containing a coffee box and multiple printed symbol cards, including an "X" marker. All other tasks in Table 3 follow the same routine, with only the instruction text and the printed targets changed. The complete routine can be summarized in the following steps, where each step explicitly references the corresponding Node (N#) and data Flow (F#) in Figure 4.

1. **Scene initialization (A1).** The simulator spawns the tabletop scene, the UR5 arm with gripper, the coffee box, and the symbol cards in Gazebo [10].

2. **Sensing and data dispatch (N1, F1, F6).** The simulated RGB-D camera (N1) publishes an RGB stream (F1) and a depth stream (F6). The RGB stream is used for VLM perception, while the depth stream is forwarded to the downstream JSON Parser & Goal Publisher Node (N5) for 3D projection.

3. **RGB preprocessing for VLM input (N2, F1, F2).** The Perception Node (N2) subscribes to the RGB topic (F1) and converts incoming messages into OpenCV frames, producing the RGB frame (F2) as the visual input to the VLM client.

4. **Instruction injection (N3, F3).** The task instruction text is provided as an external input to the system and is passed directly to the Prompt & VLM Client Node (N3) through (F3). In this example, the instruction is `move the coffee box to X`.

5. **Prompt construction and VLM request (N3, F2, F3, F4).** The Prompt & VLM Client Node (N3) combines the current RGB frame (F2) with the task instruction (F3) and a fixed prompt template. The template specifies the top-down view assumption, image resolution and pixel coordinate convention, and the required JSON output schema. The node then encodes the RGB frame and sends an HTTP request (F4) to the inference server (N4) deployed on Jetson Orin NX [11].

6. **On-device inference and JSON output (N4, F5).** The Qwen Inference Server (N4) runs Qwen 2.5-VL [8]to infer the manipulation intent and the object and target locations from the input image and prompt. It returns a compact JSON prediction (F5) containing a task type field `task` (for example `PLACE`) as well as `object\_px` and `target\_px`, which denote the center pixel coordinates of the manipulated object and the target location, respectively. In this example, the server returns: `{"task": "PLACE", "object_px": [320, 240], "target_px": [450, 180]}`.

7. **3D goal construction (N5, F5, F6, F7).** The JSON Parser & Goal Publisher Node (N5) receives the JSON output (F5) and the forwarded depth stream (F6). It validates that the predicted pixel coordinates lie inside the 640×480 image, then uses the depth values together with the camera intrinsics and the fixed camera pose to project `object_px` and `target_px` into 3D coordinates in the UR5 base frame. The resulting 3D pick and place goals are published as ROS messages (F7).

8. **Motion planning and execution (N6, F7, F8, N7).** The Motion Planning Node (N6) subscribes to the 3D goals (F7), constructs the corresponding end-effector poses, and invokes the MoveIt [53] planning pipeline. If collision-free trajectories are

found, the node sends joint trajectories and gripper commands (F8) to the simulated robot (N7). The UR5 executes a pick motion at the coffee box and places it onto the card with the "X" symbol.

9. **Success criterion.** The trial is marked as successful if the final pose of the box overlaps the target card.

In summary, the VLM component provides pixel-level semantic grounding through (N4, F5), while metric goal generation is resolved by combining this output with depth measurements in (N5, F6), and the final manipulation is executed through the motion planning and control stack (N6, F7, F8).

In the following three sections, we will introduce the entire system from the three components of Gazebo Simulation, ROS + MoveIt control stack, and Jetson Orin NX, according to the modules depicted in Figure 4.

5.3.1 Gazebo Simulation using URDF

Gazebo [10] is an open-source 3D robotics simulator that enables accurate and efficient simulation of robots in complex environments. It offers a physics engine, graphics, and a flexible API for developers to create custom simulations.

In this work, Gazebo functions as the simulation platform that unifies perception, planning, and control within a virtual environment. The scene configuration includes physics parameters, lighting, robot base, and depth camera models, providing a physically consistent workspace for robot actions. Gazebo also supplies pose information and visual data through ROS topics, enabling 3D target localization and perception tasks when combined with the vision-language model.

The Unified Robot Description Format (URDF) [65] is an XML-based file format used to describe the physical structure and properties of robots in simulation environments like Gazebo [10]. It defines a robot's geometry, links, joints, and kinematic properties, allowing developers to model complex robotic systems accurately.

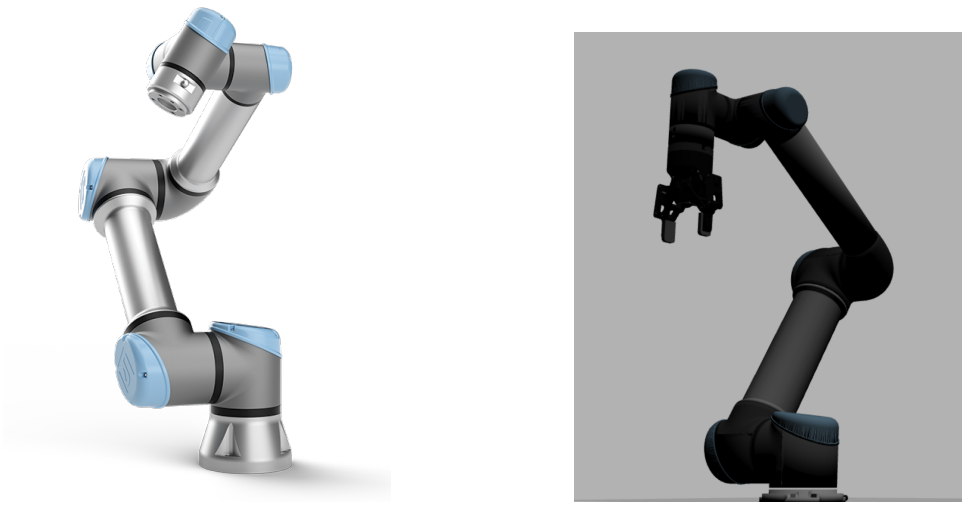
In this work, URDF is the foundational framework for modeling the robot and work scene in simulation. It specifies the robot's physical structure (links, joints, and geometries), kinematic parameters (DH table, joint limits, coordinate transformations), and dynamic properties (mass, inertia, center of mass), while also enabling seamless integration with Gazebo through visual and collision geometries. The modeling process was divided into several layers, each addressing a specific aspect of the robot description, semantics, environment, and integration. This modeling process is summarized in Table 4.

We chose the UR5 [62] as our robotic platform. It is a six-degree-of-freedom industrial arm that has become popular in both research and industry thanks to its precision, reliability, and open software ecosystem. With a 5 kg payload capacity and 850 mm reach, the UR5 works well for tabletop manipulation tasks in this work. We used the official UR5 URDF model provided by the manufacturer [66] as the basis for our simulation,

Element	Modeling method/tool	Description
Robot structure	URDF files (.urdf/.xacro)	Defines links, joints, dimensions, inertia, and joint limits
Robot semantics	SRDF files (.srdf)	Defines end-effectors, grasp groups, and planning groups
Kinematics and dynamics	MoveIt configuration package	Provides forward/inverse kinematics and collision checking
Robot appearance	Mesh models (.dae/.stl)	3D models for simulation and visualization
Workspace environment	Gazebo world files (.world)	Defines the table and static obstacles
Scene objects	Gazebo models (.sdf/.urdf)	Descriptions of target objects for manipulation
Object state publishing	Gazebo plugins, ROS topics	Real-time publication of object pose and state
Coordinate relations	ROS TF tree	Maintains spatial relations among robot, objects, and environment
System integration	ROS launch files	Unified startup of robot, environment, and planning modules

Table 4: Modeling elements of the UR5 robot arm and work scene.

specifying all links, joints, inertial parameters, and joint limits of the robot. We also generated a MoveIt [53] configuration package that handles forward/inverse kinematics, collision detection, and trajectory planning. To visualize the robot, we used 3D mesh files (`.dae`/`.stl`) for both the simulation interface and realistic rendering in Gazebo [10]. Figure 5 shows the UR5 [62] arm alongside its simulation model for comparison.



(a) The physical UR5 robotic arm. (b) UR5 model in Gazebo simulation.

Figure 5: Comparison between the physical UR5 arm and its simulated counterpart used in this work.

The work scene was modeled using world description files that defined the workspace table, static obstacles, and other relevant environmental elements. Manipulation targets

were described using Gazebo models, which allowed them to be spawned dynamically in the simulation. Object states, including positions and orientations, were continuously published through Gazebo plugins and ROS topics.

5.3.2 ROS + MoveIt control stack

The middleware layer of the system is built on ROS (Robot Operating System) [9], which provides the communication backbone for modular integration of perception, reasoning, and control components. ROS offers a publish–subscribe messaging framework, service calls, and action interfaces that enable distributed nodes to exchange information in real time. Inside the "ROS + MoveIt control stack" box in Figure 4, four ROS nodes implement the perception and control pipeline.

The Perception Node subscribes to the RGB stream from the simulated Kinect camera. It receives messages on a ROS topic and converts them to OpenCV frames. At each trial it holds the latest RGB frame in memory so that it can be passed to the vision–language model. In parallel, the depth channel is subscribed on a separate topic and is forwarded to the JSON Parser and Goal Publisher Node, where it will be used for pixel to 3D projection.

The Prompt & VLM Client Node combines visual and textual information. It receives the current RGB frame from the Perception Node and the task instruction from the evaluation script. It constructs the multimodal prompt, attaches the encoded image and sends an HTTP POST request to the Qwen [8] inference server on the Jetson.

The JSON Parser & Goal Publisher Node receives the HTTP response from Jetson as well as the forwarded depth stream. The response from Jetson is a compact JSON prediction that contains the task type and pixel coordinates for the object and the target. This node parses the JSON, validates the values and projects the pixel coordinates into 3D pick and place goals using the calibrated camera parameters and the depth from RGB-D camera. The 3D goals are then published on ROS topics for downstream planning.

The Motion Planning Node subscribes to these 3D goals. It uses the MoveIt [53] planning pipeline to compute collision free joint trajectories for the UR5 [62] arm. MoveIt is an open-source motion planning framework integrated with ROS [9], widely recognized as the leading software for robotic manipulation and kinematics. MoveIt uses libraries like Open Motion Planning Library (OMPL) [67] for sampling-based planning and supports simulators like Gazebo for efficient prototyping. For each task it generates a pick trajectory and a place trajectory and sends them as joint commands to the UR5 controllers in Gazebo. It also controls the gripper opening and closing.

5.3.3 Jetson Orin NX inference server

The A3 module in Figure 4 shows the software stack on the Jetson Orin NX [11]. The Qwen Inference Server is responsible for multimodal reasoning. At startup the server loads the Qwen 2.5-VL 3B model [8] into GPU memory and keeps it resident for the duration of the experiments.

The server is implemented as a lightweight HTTP request–response API. Each HTTP POST request from the Prompt & VLM Client Node contains an encoded image, the textual instruction and the fixed prompt template that defines the camera configuration and the desired JSON output format. The server decodes the image, and passes it to Qwen 2.5-VL [8] model. It then extracts the relevant coordinates from the generated tokens and returns a JSON object that follows the predefined schema. The JSON is sent back to the virtual machine as the HTTP response shown in Figure 4. Processing is done sequentially with a batch size of one, which matches the interactive nature of the control loop and avoids memory pressure on Jetson Orin NX [11].

5.4 Hardware Deployment

The NVIDIA Jetson Orin NX [11] serves as the primary deployment platform in this project, providing up to 100 TOPS (trillion operations per second) of AI performance within a configurable power envelope of 10W to 25W. Built on the NVIDIA Ampere architecture, it integrates a 1024-core CUDA GPU, 32 Tensor Cores, and an 8-core Arm Cortex-A78AE CPU running at up to 2 GHz, supported by 16 GB of LPDDR5 memory. With its compact form factor of 70 mm $\times$ 45 mm, the module offers a balance of compute density and energy efficiency, making it well suited for real-time multimodal reasoning and robotic control tasks under fully offline conditions. In our experiments, the Jetson's power mode was consistently set to 25W to reduce performance variability and keep runtime behavior stable during experiments.

In addition to the Jetson device, the setup includes a host machine and an Ubuntu virtual machine (VM). The host machine, a Windows laptop equipped with a discrete RTX 5060 GPU, is mainly used for development, debugging, and large-scale model testing. The Ubuntu VM, configured in bridged networking mode with a fixed IP address, runs the ROS [9] middleware, Gazebo [10] simulator, and other components required for system prototyping and validation. Development and simulation are carried out on the VM and host machine, while the Jetson Orin NX functions as the final deployment target for running the integrated inference and control framework.

6 Experiments

The following three experiments were conducted:

1. **Task Performance Evaluation** The performance of our proposed locally deployed system is evaluated using our `RT2-Sim24 Task Set`, as listed in Table 3, under simulated, resource-limited conditions.
2. **Performance - Efficiency Evaluation** Performance–efficiency trade-offs are analyzed across different model scales and deployment platforms to quantify the relationship between accuracy, responsiveness, and computational cost.
3. **Robustness Evaluation** Robustness of our proposed system is evaluated under several visual perturbations such as changes in lighting, scene complexity, and camera orientation.

6.1 Experimental Framework

The three experiments were conducted on the following hardware:

1. Jetson Orin NX for running the Qwen 2.5-VL (3B) model
2. RTX 5060 GPU for running the Qwen 2.5-VL (7B) model
3. Cloud server for running the Qwen-VL-Max model (parameters undisclosed)

Platform	**Model Variant**	**Deployment Type**
Jetson Orin NX	Qwen 2.5-VL (3B)	Local
RTX 5060 Laptop	Qwen 2.5-VL (7B)	Local
Cloud Server	Qwen-VL-Max	Online

Table 5: Model and platform configurations used for experimental evaluation.

The three tested hardware and model configurations are summarized in Table 5. All models use identical prompts, camera perspectives, and motion planning parameters to ensure a fair comparison. The Jetson platform operates fully offline with local inference, while the RTX and cloud baselines execute inference using a local PC GPU and remote API calls to a cloud server, respectively.

The 3B model on Jetson Orin NX and 7B model on RTX 5060 were executed with fixed inference parameters to ensure consistent evaluation across platforms. Key parameters and their configurations are summarized as follows:

- **DTYPE = FP16**: Half-precision floating-point computation was used to reduce memory footprint and improve inference speed on embedded hardware, while maintaining sufficient numerical stability for multimodal reasoning.
- **MAX_PIXELS = 1003520**: The maximum number of image pixels processed per forward pass, corresponding to a resolution of approximately 1280×784. This setting balances visual detail preservation and computational efficiency.

- **DEFAULT_NUM_BEAMS = 1**: Beam search was disabled to avoid excessive decoding latency on Jetson Orin NX. A single-beam (greedy) decoding strategy was adopted for faster and more deterministic output generation.
- **DEFAULT_DO_SAMPLE = true**: Sampling was enabled to introduce controlled stochasticity during decoding, preventing repetitive or overly rigid output sequences.
- **DEFAULT_TEMPERATURE = 0.3**: A low temperature value was chosen to ensure stable and structured responses, particularly for JSON-formatted outputs, while retaining moderate diversity.
- **DEFAULT_TOP_P = 0.9**: Nucleus (top-p) sampling was used to filter out low-probability tokens and improve decoding efficiency. This threshold retains the top 90% cumulative probability mass for generation.
- **MAX_NEW_TOKENS = 256**: The maximum number of tokens allowed per response. This limit ensures that the model can generate a complete JSON instruction or textual explanation without truncation (the original default of 128 tokens was insufficient).

For **Experiment 1**, all models were tested using our `RT2-Sim24 Task Set`, covering three semantic categories: `Symbol Understanding`, `Reasoning`, and `Person Recognition`. Each task was repeated five times, resulting in a total of 120 trials per model.

Baseline configuration. The primary task performance evaluation was conducted under a clean and fully controlled simulation environment, where all perceptual and physical settings were fixed:

- Fixed object positions, identical across repetitions
- Fixed gripper parameters (contact stiffness, contact damping, friction coefficient)
- Fixed camera pose (no tilt, nominal intrinsics/extrinsics)

Extended evaluation conditions. To assess how well the system generalizes beyond the idealized baseline, Experiment 1 further introduced three controlled perturbations. In each condition, only one factor was varied while all others remain fixed, allowing us to isolate its effect on task success. Across all extended settings, every task in our `RT2-Sim24 Task Set` was repeated five times, as the same with baseline configuration. The three extended settings were:

1. **Randomized object positions** The target object was randomly placed within a bounded workspace region (within 5cm offset in the x/y directions relative to the fixed position).
2. **Randomized gripper physical parameters** The stiffness, damping, and friction coefficients of the gripper were randomly perturbed within ±20% of the fixed values.
3. **Randomized camera pose** The camera pitch was randomly perturbed within ±10°, modifying the viewpoint while leaving the scene unchanged.

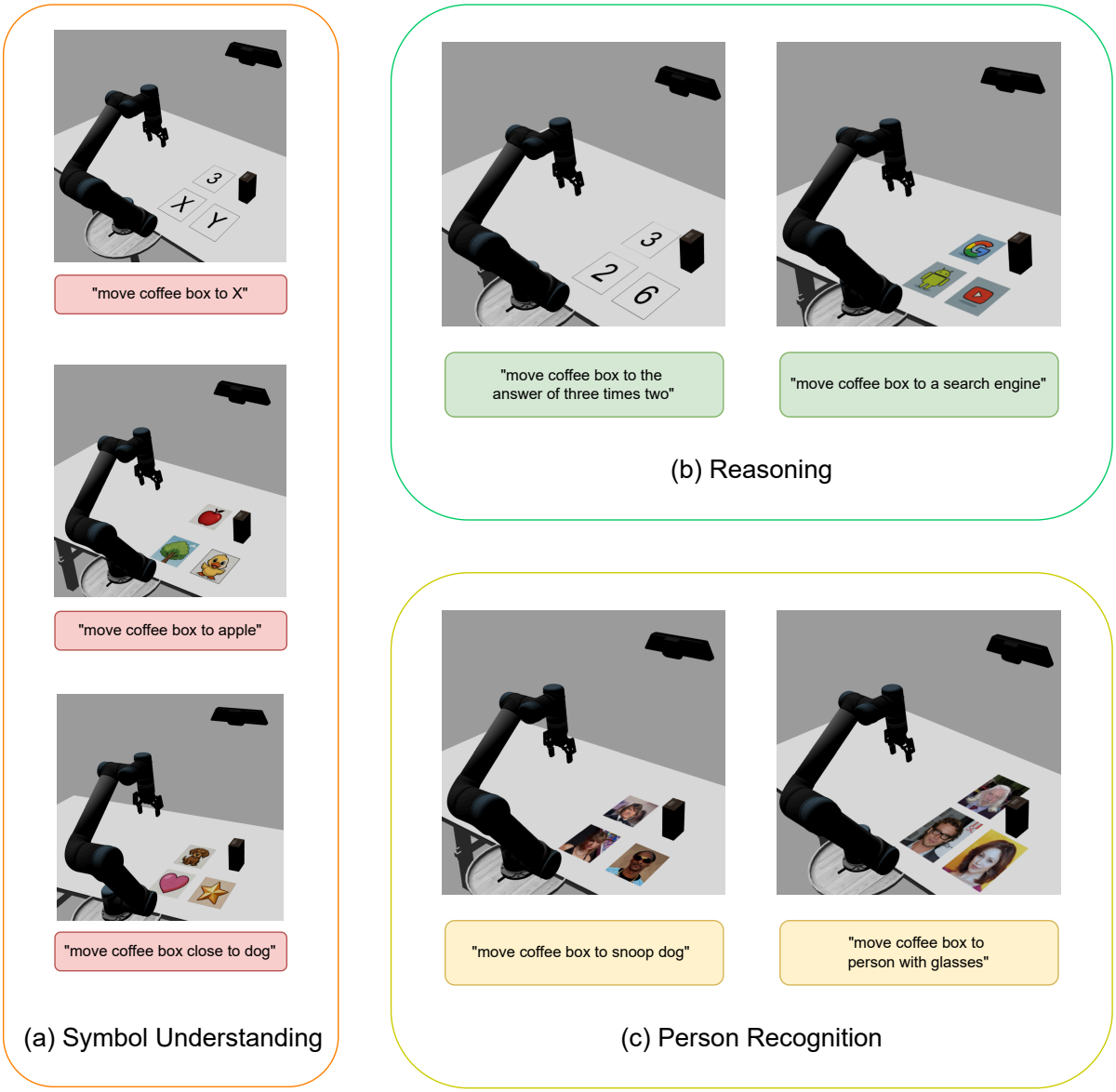
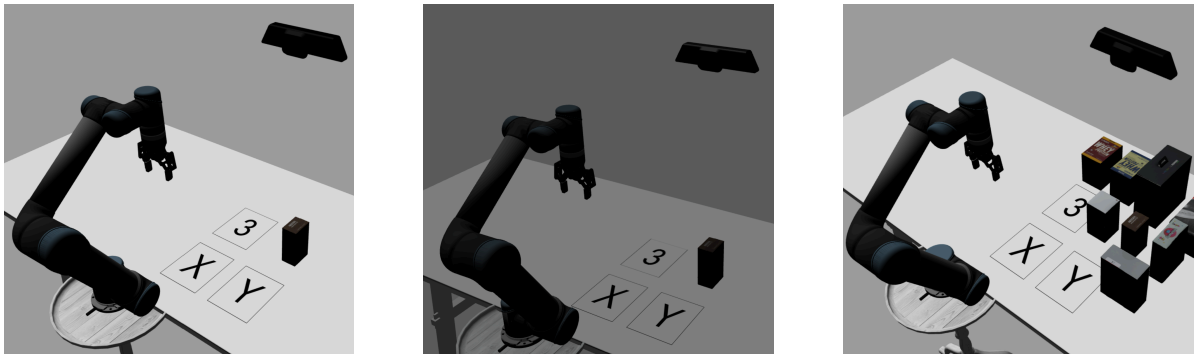


Figure 6: Representative task scenarios used in Experiments 1 and 2. Our `RT2-Sim24 Task Set` was employed for both success rate and latency evaluations across all model configurations.

For **Experiment 2**, similar to the baseline setting of Experiment 1 (clean scene, fixed pose, fixed positions), all models were evaluated on our `RT2-Sim24 Task Set`, with 5 repetitions per task (120 trials per model). All task scenes in Experiment 1 and 2 are illustrated in Figure 6.

For **Experiment 3**, we used the `Symbol Understanding` subset of `RT2-Sim24 Task Set`, which contains 9 language-conditioned pick-and-place tasks to focus on the visual robustness without confounding task variability. Each task was repeated 5 times under each visual condition (clean scene, lighting variation, and cluttered background), resulting in a total of 45 trials from which the success rate was computed. Figure 7 shows the respective scenes.



(a) clean scene (b) lighting change (c) cluttered background

Figure 7: Visual environments used for Experiment 3, robustness evaluation: (a) clean scene, (b) lighting change, and (c) cluttered background. These setups simulate different levels of perceptual perturbation in the robot's workspace.

6.2 Evaluation Metrics

In our evaluations we use two metrics: *task success rate* and *latency* (Inference latency and End-to-End latency in sec).

The *task success rate* (SR) is defined in Equation 1. Following the definition in RT-2 [4], for each language-conditioned manipulation task, a trial is considered successful if the robot correctly grasps the target object specified by the instruction and places it at the intended location without collision or dropping. Consistent with this protocol [4], we rely on a binary success metric assessed by human evaluators and do not measure location deviation. The *task success rate* (SR) is computed as the ratio of successful trials to the total number of executed trials:

$$SR = \frac{N_{\text{success}}}{N_{\text{total}}} \times 100\%. \tag{1}$$

Each task is repeated five times per condition, and the average SR across tasks is reported for each configuration.

To evaluate the *latency* of our multimodal reasoning and control system, the following two types of latency were measured:

- **Inference latency.**
  This metric captures the time (in seconds) consumed solely by the vision-language model to process an image–text query and generate a structured response. It is measured from the moment the model receives the visual–linguistic input until the completion of its textual or JSON output. This latency reflects the intrinsic computational efficiency of the vision–language model under the different deployment configurations as listed in Table 5.
- **End-to-End latency**
  This metric represents the total time (in seconds) from the issuance of a natural-language command to the completion of the corresponding action by the robot arm. Specifically, timing starts when the system receives the user's instruction and ends when the motion execution node publishes the `/motion_done` signal, indicating that the arm has finished both pick and place actions and returned to its initial pose. This latency includes all intermediate stages: visual frame capture, image encoding, VLM inference, JSON parsing, coordinate publication, motion planning, and trajectory execution. Therefore, End-to-End latency reflects the overall responsiveness and real-time performance of the entire embodied control loop of our proposed system on the selected compute platforms.

Both latencies were automatically logged during each task execution, enabling a comparison of computational performance across the selected hardware configurations and communication settings.

7 Experimental Results

This section presents the experimental results organized around the three research questions formulated in Chapter 1.

7.1 Experiment 1: Task Performance

Table 6 reports the task success rates across all baseline and extended conditions. The first two columns correspond to the reference RT-2 [4] models. As discussed in Section 4.5, RT-2 numbers are real-world results and are shown only as semantic reference; our results are from simulation. The third column presents the performance of the Jetson-based Qwen 2.5-VL (3B) [8] model under the controlled baseline setting, which consists of a clean scene, fixed object positions, a fixed camera pose, and fixed gripper parameters. The last three columns represent the results obtained under the extended evaluation conditions, where one system component was varied at a time while all other components remained unchanged.

Task Category	Real World Experiments		Simulation on `RT2-Sim24 Task Set`			
			Qwen 2.5-VL (3B)			
	RT-2 PaLI-X (55B)	RT-2 PaLM-E (12B)	Baseline	Random Object Position	Random Gripper Parameters	Random Camera Pose
Symbol Understanding	82	36	93	87	80	51
Reasoning	46	43	98	89	73	51
Person Recognition	53	43	87	77	77	27
Average	**60**	**40**	**93**	**85**	**77**	**45**

Table 6: The first two columns show the task success rates (%) for two RT-2 [4] models on the real world task set described in Section 4.4. The results of our proposed system on our `RT2-Sim24 Task Set` are listed in the remaining columns, including the Jetson-based baseline setting, and three extended evaluation conditions. The extended conditions vary a single factor at a time while keeping all other components fixed (See Section 6.1). Note that the RT-2 experiments took place in real world, while Qwen [8] experiments were conducted in simulation.

Several patterns can be identified from the results. First, under the controlled baseline setting, the Jetson-deployed Qwen 2.5-VL (3B) model achieves an average success rate of 93%. As discussed in Section 4.5, clearly a direct comparison between the results obtained using a real-world setup and our results in a simulated setting is not possible, but it could be noted that this value is higher than the success rates reported for RT-2 PaLI-X and RT-2 PaLM-E. Among the three task categories, the highest performance is observed in the Reasoning tasks, followed by Symbol Understanding, while Person Recognition remains the most challenging.

Second, the extended experiments show the impact of varying different components of the perception and control pipeline. When the object positions are randomized, the average success rate decreases to 85%, which indicates that the system remains stable when the spatial arrangement deviates from the original configuration. When the physical parameters of the gripper are perturbed, the success rate decreases further to 77%. This result suggests that variation in contact dynamics influences execution stability more strongly than variation in object layout. The largest performance reduction occurs when the camera pose is altered within a 10 degree tilt range. In this condition the average success rate decreases to 45%, and the drop is especially pronounced in the Person Recognition tasks. These results show that the visual encoder is sensitive to viewpoint changes.

7.2 Experiment 2: Performance–Efficiency

We evaluated the task success rate and latency of the proposed multimodal robotic system under three model configurations: `Qwen 2.5-VL (3B)` running fully offline on the Jetson Orin NX, `Qwen 2.5-VL (7B)` on an RTX 5060 GPU, and `Qwen-VL-Max` model on a cloud server. All models were evaluated on our `RT2-Sim24 Task Set`.

As shown in Figure 8, the Jetson-based `Qwen 2.5-VL (3B)` achieved an average success rate of 93%, closely approaching the RTX-based `7B` variant (96%) and the cloud `Qwen -VL-Max` model (100%). Across all categories, the 3B model exhibited strong semantic grounding and reasoning ability: it reached 93% in `Symbol Understanding` and 98% in `Reasoning`, only marginally lower than the much larger and cloud-hosted counterparts. The most notable performance gap appeared in the `Person Recognition` category, where fine-grained visual identification is inherently more demanding. Here, the 3B model achieved 87%, compared with 90% for the 7B model and 100% for the cloud model.

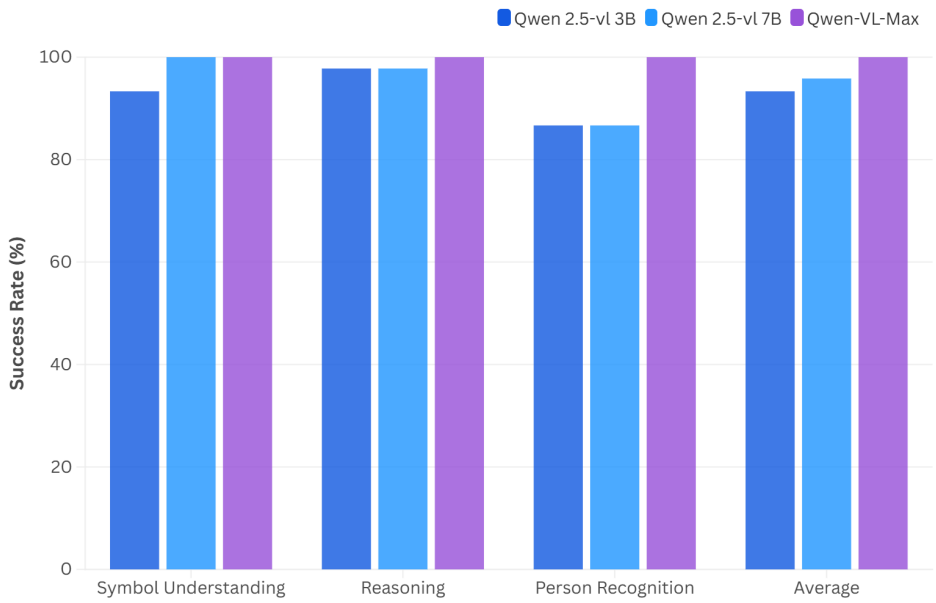


Figure 8: Comparison of task success rates between three model configurations on RT2-Sim24 Task Set.

For latency, as shown in Table 7 and Figure 9, the Jetson-deployed `Qwen 2.5-VL (3B)` model achieved an average inference latency of 10.06 s, which is approximately 2.9× lower than the larger 7B variant running on the RTX 5060 (29.32 s), and about 2.3× slower than the cloud-based model (4.30 s). When considering the full perception-reasoning-control pipeline, the Jetson system reached an end-to-end latency of 32.23 s, closely approaching the cloud configuration (28.30 s) and significantly outperforming the laptop setup (57.81 s).

Model Configuration	Inference Latency (s)	E2E Latency (s)	Average SR (%)
Qwen 2.5-VL (3B, Jetson Orin NX)	10.06	32.23	93
Qwen 2.5-VL (7B, RTX 5060)	29.32	57.81	96
Qwen-VL-Max (Cloud)	4.30	28.30	100

Table 7: Comparison of Inference Latency, End-to-End Latency, and Average Success Rate for the three model and platform configurations.

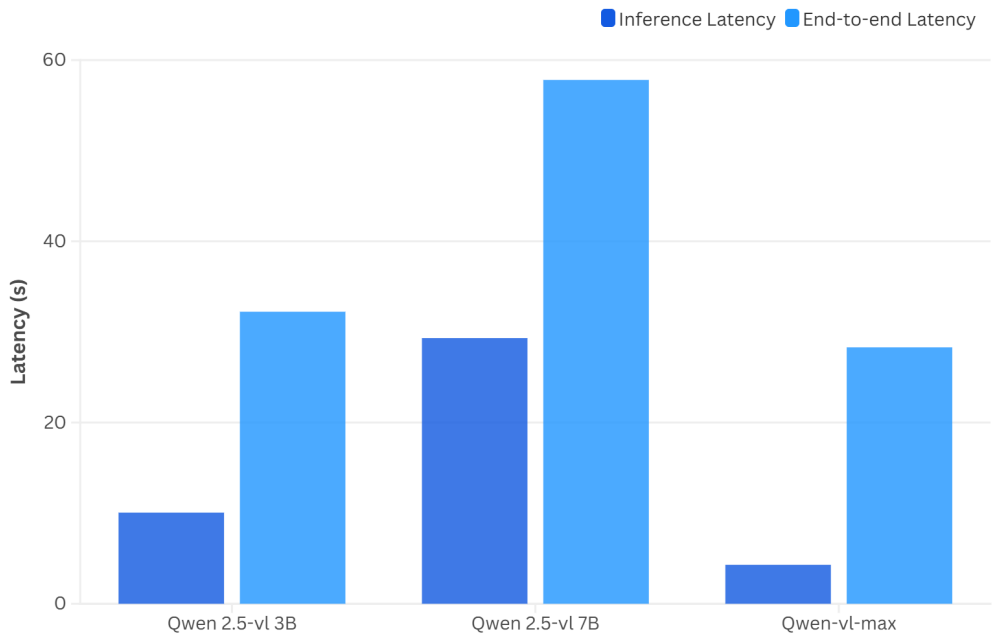


Figure 9: Comparison of Inference and End-to-End Latency across three model and platform configurations.

7.3 Experiment 3: Robustness

To evaluate robustness of our proposed system, we separate two sources of variation. The three extended evaluations in **Experiment 1** perturbs system and sensing configurations (e.g., object layout, gripper parameters, and camera viewpoint). **Experiment 3** focuses on environment-driven visual disturbances on Jetson Orin NX, and runs the same tasks under three scene settings:

- a clean scene with no visual interference
- a mild perturbation condition involving changes in lighting intensity and direction
- a cluttered background condition where multiple distractor objects were added to the workspace

Figure 7 illustrates the three visual environments used in the robustness evaluation, corresponding to the clean, lighting-change, and cluttered-background settings. These scenarios simulate typical sources of perceptual noise in real-world table-top manipulation, allowing us to assess whether our multimodal reasoning system remains reliable when the visual input changes.

As shown in Table 8, the Jetson-deployed Qwen 2.5-VL (3B) model achieved a 93% success rate in the clean scene. Under mild lighting changes, performance slightly increased to 96% (+3% relative), indicating that moderate illumination variation did not degrade perception in our setup. In contrast, the cluttered background condition caused a substantial drop to 53% (-40% relative), suggesting that additional distractor objects introduce significant visual ambiguity for object–target grounding in the current pipeline. Overall, our proposed system remains stable under lighting perturbations but is strongly affected by background clutter in this evaluation.

Visual Condition	**Success Rate (%)**	**Change (%)**
Clean scene	93	–
Mild perturbation (lighting change)	96	+3
Cluttered background	53	−40

Table 8: Task success rate (%) of the Jetson-deployed Qwen 2.5-VL (3B) model under different visual perturbation settings. Each condition was evaluated on the same 45 trials derived from the `Symbol Understanding` subset of `RT2-Sim24 Task Set`.

8 Discussion

This chapter provides an integrative discussion of the overall system design and the experimental results presented in previous sections. It synthesizes the main findings from Chapter 7, highlighting how the proposed framework demonstrates the feasibility of deploying a multimodal large model for language-conditioned manipulation on an embedded platform. Beyond quantitative outcomes, this chapter reflects on the distinctive characteristics of the implemented system. The chapter concludes by acknowledging current limitations and identifying promising directions for future research.

8.1 Summary of Experimental Findings

This study investigated the feasibility of executing embodied multimodal reasoning and control entirely on resource-constrained edge hardware. The evaluation consisted of three groups of simulation experiments that correspond to the research questions on feasibility, efficiency and robustness described in Chapter 1.

Feasibility. As discussed in Section 4.5, our experiments are conducted in a Gazebo [10] simulation and therefore cannot be directly compared to the real-world RT-2 [4] evaluations. For this reason, even though the Jetson-deployed Qwen 2.5-VL (3B) [8] model reaches an average success rate of 93% on our `RT2-Sim24 Task Set`, this number must not be interpreted as evidence of superiority over RT-2-PaLI-X and RT-2-PaLM-E. The RT-2 results were obtained in real physical environments and therefore include sensing noise, calibration inaccuracy, contact uncertainty and execution variability. In contrast, our experiments were conducted in simulation, where these disturbances are substantially reduced. Despite this, the results show that, within a controlled simulation, a compact model deployed on embedded system can reliably execute language-conditioned manipulation tasks instantiated in our `RT2-Sim24 Task Set`.

It is also important to interpret the results by task category, since the distribution of success rates across task categories differs across models. Even within RT-2, the distribution across task categories is not consistent: PaLI-X achieves 82% on Symbol Understanding but only 46% on Reasoning, while PaLM-E shows a much flatter profile (36–43%) and a different ordering. The RT-2 paper [4] does not provide a detailed explanation for these differences between the variants. However, one possible explanation is that category-wise performance is influenced by model-specific factors, such as model capacity and the vision–language representations learned during training, which may favor certain capabilities over others. Our compact 3B model shows a different distribution on `RT2-Sim24` (Reasoning 98%, Symbol Understanding 93%, Person Recognition 87%), which further shows that the distribution of success rates across task categories can differ across models.

The three extended experiments further clarify the scope of feasibility. When object positions were randomized, the success rate decreased moderately to 85%, which shows that the system maintains semantic grounding under changes in spatial layout. When the physical parameters of the gripper were perturbed, the success rate decreased to 77%. This suggests that modifications related to execution dynamics influence task outcomes more strongly than variation in object arrangement. The most significant drop occurred

under camera-pose tilting of up to 10 degrees, where the success rate fell to 45%. This finding reveals a substantial viewpoint sensitivity in the small vision-language model, even in simulation. Since real robotic systems experience far greater variation in camera pose, illumination and extrinsic calibration, this sensitivity represents a major feasibility constraint that must be addressed before moving from simulation to physical deployment.

Performance–efficiency. When we compared platforms, we found that bigger models (3B→7B→cloud) only improved accuracy slightly, while their latency and resource demands increased substantially. The Jetson configuration achieved ≈93% accuracy with 2.9× lower inference latency than the 7B RTX setup. These findings show that the Jetson-based 3B model provides a favorable balance between resource usage, responsiveness and semantic task performance within simulation. Larger models deliver somewhat higher accuracy but suffer from sharply increased latency and hardware requirements. The cloud model is the fastest in raw inference. However, in our setup, end-to-end latency does not improve proportionally because it additionally depends on network and API overhead, whereas edge execution avoids such variability.

Robustness. Robustness of our proposed system was evaluated from two complementary perspectives: (i) sensitivity to single-factor perturbations in the system configuration (Experiment 1), and (ii) tolerance to environmental visual disturbances (Experiment 3), both on the Jetson-based Qwen 2.5-VL (3B) pipeline in simulation. In Experiment 1, success rate decreased from 93% in the baseline to 85% with random object positions, 77% with random gripper parameters, and 45% with random camera pose, while keeping the rest of the pipeline fixed. In Experiment 3, the system achieved a 93% success rate in the clean scene and remained stable under mild lighting change, slightly increasing to 96%. In contrast, performance dropped sharply to 53% under a cluttered background.

These findings point to two dominant robustness bottlenecks: sensitivity to viewpoint variation and sensitivity to background clutter, both of which directly affect visual grounding before motion execution. It is important to note that the simulated disturbances remain simpler than those encountered in real environments. Real lighting introduces complex shadows, reflections, and exposure changes, and real clutter often causes occlusions and texture overlap. Therefore, the robustness results should be interpreted as robustness within a controlled simulation environment, rather than evidence of real-world robustness.

Collectively, these three groups of simulation experiments show that fully offline multimodal reasoning and control is feasible on compact embedded hardware in a controlled setting. The Jetson-deployed model achieves low-latency on-device inference and high baseline success in simulation, but performance drops markedly under cluttered backgrounds and camera viewpoint changes. This work provides a practical starting point for language-driven manipulation on embedded systems, while highlighting the specific conditions that must be addressed to reduce the simulation-to-real gap.

8.2 Key System Features

The proposed system adopts a modular architecture in which perception, reasoning, and control components are independently developed and integrated through standardized

interfaces. Each module communicates via well-defined data formats such as ROS [9] topics or structured JSON outputs. This modularity allows components to be replaced or upgraded with minimal system modification, ensuring flexibility and long-term maintainability.

A second feature is fully offline autonomy. All components, including the vision-language model, operate locally on the Jetson Orin NX [11] without relying on external network resources. This design guarantees reliable execution in isolated environments and reduces latency introduced by cloud communication, enabling stable execution without network dependency.

The system also follows a simulation-driven development paradigm. Using Gazebo [10], URDF [65], and MoveIt [53], the entire perception–planning–execution loop can be validated in a realistic 3D environment before deployment. This workflow helps reduce integration risk before physical deployment and supports more consistent interfaces between simulation and real setups.

8.3 Limitations

We summarize the main limitations of our proposed system and evaluation as follows:

- **Edge inference constraints.** Although Jetson Orin NX [11] supports CUDA acceleration and mixed-precision inference, its limited GPU memory and compute restrict the deployment of larger multimodal models and higher input resolutions. As a result, the system operates under a speed-accuracy trade-off, and some tasks still show latency that is higher than desired for responsive manipulation.
- **Simulation-to-real gap.** All evaluations are conducted in Gazebo [10]. Simulation does not capture real sensing and execution uncertainty, and the system is sensitive to changes such as camera pose and clutter background. Therefore, results cannot be directly compared to RT-2 real-world results [4].
- **Limited language flexibility.** The pipeline relies on structured prompts and JSON outputs, which works well for fixed templates but is less stable for natural or ambiguous instructions. Multi-turn clarification and intent disambiguation are not supported.

8.4 Future Work

Future work will focus on several directions that aim to extend the capabilities of the current system and reduce the gap between simulation and real-world operation:

- **Improving system scalability and automation.** In the current implementation, each module was manually configured and tuned, which constrained reproducibility and rapid testing. Enhancing configuration management, automatic calibration, and integrated logging will help streamline deployment and debugging. These improvements will also allow the system to adapt more efficiently to changes in hardware configuration and experimental environments.

- **Optimization of the multimodal model for embedded deployment.** The present implementation of Qwen 2.5-VL [8] achieves stable inference under mixed-precision settings, but its computational cost remains significant. Future work will investigate model compression, weight quantization, and knowledge distillation techniques to reduce memory footprint and latency without sacrificing task performance. Incorporating hardware-aware scheduling and adaptive batching strategies may further improve real-time responsiveness under varying workloads.
- **Explore more flexible task understanding and interaction mechanisms.** Current language interface follows a structured schema, which limits its ability to interpret free-form instructions. Integrating dialogue-based reasoning and incremental scene understanding would allow the robot to handle multi-step and ambiguous commands. These extensions, combined with sim-to-real adaptation and system automation, will transform the current prototype into a deployable and resilient multimodal robotic platform for real-world applications.
- **Transfer from simulation to the real robotic platform.** Although the Gazebo [10] environment provides a controlled and convenient setting for prototyping, it cannot reproduce the full range of physical uncertainties, sensor noise, or actuation variability present in real environments. The next step will therefore involve validating the framework on an actual robot arm in order to assess its performance under real-world conditions. In particular, the simulated setup mirrors a standard real tabletop configuration with a UR5 [62] robot arm and a Kinect RGB-D camera, both of which are available as physical devices. This makes it practical to transfer the same pipeline to a real setup and quantify how the system behaves under real sensing, calibration, and control uncertainty.

9 Conclusion

This thesis investigated the feasibility of executing embodied multimodal reasoning and manipulation entirely offline on resource-constrained edge hardware. We proposed a modular Edge-VLA system that integrates perception, on-device VLM inference, and ROS-MoveIt-based robot motion execution for tabletop pick-and-place. To the best of our knowledge, this work is among the first to demonstrate a fully offline vision-language-action manipulation pipeline on an embedded Jetson device using a vision-language model.

Experiments in simulation show that an embedded deployment is practically viable: the Jetson-deployed Qwen 2.5-VL (3B) achieves a 93% success rate on our `RT2-Sim24 Task Set`. Robustness evaluations indicate that performance is relatively stable under moderate perturbations, while two factors dominate failure: heavy visual clutter (53%) and camera pose tilts up to $\pm 10^{\circ}$ (45%). These results suggest that fully offline embodied manipulation on embedded hardware is feasible in controlled settings, and that improving viewpoint robustness and clutter tolerance will be important for narrowing the gap toward real-world deployment.

The main contributions of this thesis are summarized as follows:

1. **Feasibility of offline embodied reasoning.** We implement an end-to-end pipeline on an embedded device, suggesting that vision-language reasoning and robot control can be practically executed without network dependency.

2. **Empirical validation on the `RT2-Sim24 Task Set`.** We conducted three simulation experiments: (1) task performance under baseline and single-factor perturbations, (2) performance-efficiency trade-offs across Jetson/RTX/cloud deployments, and (3) robustness under visual appearance shifts.

3. **Modular and reproducible architecture.** We implement perception, reasoning, and robot control within a unified framework named Edge-VLA, providing a reproducible and extensible foundation for future work on resource-efficient embodied AI and sim-to-real adaptation.

References

[1] G. Pezzulo, T. Parr, P. Cisek, A. Clark, and K. Friston, "Generating meaning: Active inference and the scope and limits of passive ai," *Trends in Cognitive Sciences*, vol. 28, no. 2, pp. 97–112, 2024.

[2] D. Driess et al., "PaLM-E: An embodied multimodal language model," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 8469–8488, 2023.

[3] Y. Ma, Z. Song, Y. Zhuang, J. Hao, and I. King, "A survey on vision-language-action models for embodied ai," *arXiv preprint arXiv:2405.14093*, 2024.

[4] B. Zitkovich et al., "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning (CoRL)*, PMLR, pp. 2165–2183, 2023.

[5] M. J. Kim et al., "OpenVLA: An open-source Vision-Language-Action model," in *Conference on Robot Learning (CoRL)*, PMLR, pp. 2679–2713, 2025.

[6] K. Black et al., "π_0: A vision-language-action flow model for general robot control," *arXiv preprint arXiv:2410.24164*, 2024.

[7] X. Chen et al., "PaLI-X: On scaling up a multilingual vision and language model," *arXiv preprint arXiv:2305.18565*, 2023.

[8] S. Bai et al., "Qwen2.5-vl technical report," *arXiv preprint arXiv:2502.13923*, 2025.

[9] Open Robotics, *ROS: Robot Operating System*. [Online]. Available: `https://www.ros.org/`, 2025.

[10] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, vol. 3, pp. 2149–2154, 2004.

[11] NVIDIA, "Jetson Orin NX series data sheet," NVIDIA Corporation, Tech. Rep. [Online]. Available: `https://developer.nvidia.com/downloads/jetson-orin-nx-series-data-sheet`, 2023.

[12] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3128–3137, 2015.

[13] J. Achiam et al., "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[14] Y. Kim, D. Kim, J. Choi, J. Park, N. Oh, and D. Park, "A survey on integration of large language models with intelligent robots," *Intelligent Service Robotics*, vol. 17, no. 5, pp. 1091–1107, 2024.

[15] J. Wei et al., "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[16] A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

[18] L. Guan, S. Sreedharan, K. Valmeekam, and S. Kambhampati, "Leveraging pre-trained large language models to construct and utilize world models for model-based task planning," in *37th Conference on Neural Information Processing Systems, NeurIPS 2023*, Neural information processing systems foundation, 2023.

[19] A. Radford et al., "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 8748–8763, 2021.

[20] A. Guzhov, F. Raue, J. Hees, and A. Dengel, "AudioCLIP: Extending clip to image, text and audio," in *2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, pp. 976–980, 2022.

[21] J. Yang et al., "LLM-Grounder: Open-vocabulary 3d visual grounding with large language model as an agent," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 7694–7701, 2024.

[22] C. Huang, O. Mees, A. Zeng, and W. Burgard, "Audio visual language maps for robot navigation," in *International Symposium on Experimental Robotics*, Springer, pp. 105–117, 2023.

[23] W. Khoo et al., "Spill the tea: When robot conversation agents support well-being for older adults," in *Companion of the 2023 ACM/IEEE international conference on human-robot interaction*, pp. 178–182, 2023.

[24] Q. Gu et al., "ConceptGraphs: Open-vocabulary 3d scene graphs for perception and planning," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 5021–5028, 2024.

[25] C. H. Song, B. M. Sadler, J. Wu, W.-L. Chao, C. Washington, and Y. Su, "LLM-Planner: Few-shot grounded planning for embodied agents with large language models," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, IEEE, pp. 2986–2997, 2023.

[26] Y. Dai, R. Peng, S. Li, and J. Chai, "Think, act, and ask: Open-world interactive personalized robot navigation," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 3296–3303, 2024.

[27] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3-4, pp. 189–208, 1971.

[28] B. Liu et al., "LLM+P: Empowering large language models with optimal planning proficiency," *arXiv preprint arXiv:2304.11477*, 2023.

[29] A. Brohan et al., "RT-1: Robotics transformer for real-world control at scale," *Robotics: Science and Systems XIX*, 2023.

[30] Y. Du et al., "Guiding pretraining in reinforcement learning with large language models," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 8657–8677, 2023.

[31] J.-B. Alayrac et al., "Flamingo: A visual language model for few-shot learning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 35, pp. 23 716–23 736, 2022.

[32] C. Schuhmann et al., "LAION-400M: Open dataset of clip-filtered 400 million image-text pairs," *arXiv preprint arXiv:2111.02114*, 2021.

[33] J. Li, D. Li, C. Xiong, and S. Hoi, "BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 12 888–12 900, 2022.

[34] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, pp. 34 892–34 916, 2023.

[35] B. Lin et al., "MoE-LLaVA: Mixture of experts for large vision-language models," *arXiv preprint arXiv:2401.15947*, 2024.

[36] R. Li, Z. Zhang, C. He, Z. Ma, V. M. Patel, and L. Zhang, "Dense multimodal alignment for open-vocabulary 3d scene understanding," in *European Conference on Computer Vision (ECCV)*, Springer, pp. 416–434, 2024.

[37] H. Zhen et al., "3D-VLA: A 3d vision-language-action generative world model," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 61 229–61 245, 2024.

[38] K. Nguyen, D. Dey, C. Brockett, and B. Dolan, "Vision-based navigation with language-based assistance via imitation learning with indirect intervention," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, pp. 12 519–12 529, 2019.

[39] OpenAI. "GPT-4o-mini model page. "[Online]. Available: `https://platform.openai.com/docs/models/gpt-4o-mini`, 2025.

[40] P. Wang et al., "Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution," *arXiv preprint arXiv:2409.12191*, 2024.

[41] Z. Chen et al., "Expanding performance boundaries of open-source multimodal models with model, data, and test-time scaling," *arXiv preprint arXiv:2412.05271*, 2024.

[42] Alibaba. "Qwen2.5-vl: Next-generation vision language model. "[Online]. Available: `https://qwen.ai/blog?id=qwen2.5-vl`, 2025.

[43] A. Dosovitskiy, "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[44] H. Touvron et al., "LLaMA: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[45] S. Reed et al., "A generalist agent," *Transactions on Machine Learning Research*, 2022.

[46] Y. Jiang et al., "VIMA: General robot manipulation with multimodal prompts," in *NeurIPS 2022 Foundation Models for Decision Making Workshop*, 2022.

[47] D. Ghosh et al., "Octo: An open-source generalist robot policy," in *Robotics: Science and Systems*, 2024.

[48] M. Ahn et al., "Do as i can, not as i say: Grounding language in robotic affordances," *arXiv preprint arXiv:2204.01691*, 2022.

[49] W. Huang, C. Wang, R. Zhang, Y. Li, J. Wu, and L. Fei-Fei, "VoxPoser: Composable 3d value maps for robotic manipulation with language models," in *Conference on Robot Learning (CoRL)*, PMLR, pp. 540–562, 2023.

[50] J. Lin, H. Yin, W. Ping, P. Molchanov, M. Shoeybi, and S. Han, "VILA: On pre-training for visual language models," in *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, pp. 26 679–26 689, 2024.

[51] H. Huang, F. Lin, Y. Hu, S. Wang, and Y. Gao, "CoPa: General robotic manipulation through spatial constraints of parts with foundation models," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 9488–9495, 2024.

[52] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "ReKep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," in *Conference on Robot Learning (CoRL)*, PMLR, pp. 4573–4602, 2025.

[53] MoveIt Community, *MoveIt: Open-source motion planning framework*. [Online]. Available: `https://moveit.picknik.ai/main/index.html`, 2025.

[54] A. Turing, "Computing machinery and intelligence.," *Mind*, 1950.

[55] Y. Hu et al., "Toward general-purpose robots via foundation models: A survey and meta-analysis," *arXiv preprint arXiv:2312.08782*, 2023.

[56] J. Li, D. Li, S. Savarese, and S. Hoi, "BLIP-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *International Conference on Machine Learning (ICML)*, PMLR, pp. 19 730–19 742, 2023.

[57] Z. Zhu et al., "Is sora a world simulator? a comprehensive survey on general world models and beyond," *arXiv preprint arXiv:2405.03520*, 2024.

[58] W. Su, L. Li, F. Liu, M. He, and X. Liang, "AI on the edge: A comprehensive review," *Artificial Intelligence Review*, vol. 55, no. 8, pp. 6125–6183, 2022.

[59] R. Singh and S. S. Gill, "Edge AI: A survey," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 71–92, 2023.

[60] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha, "A systematic survey of prompt engineering in large language models: Techniques and applications," *arXiv preprint arXiv:2402.07927*, 2024.

[61] G. Marvin, N. Hellen, D. Jjingo, and J. Nakatumba-Nabende, "Prompt engineering in large language models," in *International conference on data intelligence and cognitive informatics*, Springer, pp. 387–402, 2023.

[62] Universal Robots, "UR5 Technical Specifications," Universal Robots, Tech. Rep. [Online]. Available: `https://www.universal-robots.com/media/50588/ur5_en.pdf`, 2016.

[63] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2015.

[64] NVIDIA, *JetPack SDK*. [Online]. Available: `https://developer.nvidia.com/embedded/jetpack`, 2025.

[65] ROS, *URDF*, Open Source Robotics Foundation. [Online]. Available: `https://wiki.ros.org/urdf`, 2025.

[66] Universal Robots, *Universal Robots ROS2 Description*. [Online]. Available: `https://github.com/UniversalRobots/Universal_Robots_ROS2_Description`, 2025.

[67] Kavraki Lab, *OMPL: Open motion planning library*. [Online]. Available: `https://ompl.kavrakilab.org/`, 2025.

A Supplementary Tables

Task Group	Tasks
Unseen Objects (Easy)	pick banana, move banana near coke can, move orange can near banana, pick oreo, move oreo near apple, move redbull can near oreo, pick pear, pick coconut water, move pear near coconut water, move pepsi can near pear
Unseen Objects (Hard)	pick cold brew can, pick large orange plate, pick chew toy, pick large tennis ball, pick bird ornament, pick fish toy, pick ginger lemon kombucha, pick egg separator, pick wrist watch, pick green sprite can, pick blue microfiber cloth, pick yellow pear, pick pretzel chip bag, pick disinfectant wipes, pick pineapple hint water, pick green cup, pick pickle snack, pick small blue plate, pick small orange rolling pin, pick octopus toy, pick catnip toy
Unseen Backgrounds (Easy)	pick green jalapeno chip bag, pick orange can, pick pepsi can, pick 7up can, pick apple, pick blue chip bag, pick orange, pick 7up can, move orange near sink, pick coke can, pick sponge, pick rxbar blueberry
Unseen Backgrounds (Hard)	pick wrist watch, pick egg separator, pick green sprite can, pick blue microfiber cloth, pick yellow pear, pick pretzel chip bag, pick disinfectant wipes, pick pineapple hint water, pick green cup, pick pickle snack, pick small blue plate, pick small orange rolling pin, pick octopus toy, pick catnip toy, pick swedish fish bag, pick large green rolling pin, pick black sunglasses
Unseen Environments (Easy)	pick coke can, pick apple, pick rxbar blueberry, move apple near coke can, move rxbar blueberry near apple, move coke can near rxbar blueberry, pick blue plastic bottle, pick sponge, pick blue chip bag, move sponge near blue plastic bottle, move blue chip bag near sponge, move blue plastic bottle near blue chip bag, move coke can near white mug, move sponge near white mug, move coke can near yellow bowl, move sponge near yellow bowl, move coke can near green cloth, move sponge near green cloth, move coke can near plate, move sponge near plate, move coke can near spoon, move sponge near spoon, move coke can near orange cup, move sponge near orange cup, pick white mug, pick yellow bowl, pick green cloth, move white mug near sponge, move yellow bowl near sponge, move green cloth near sponge, pick plate, pick spoon, pick orange cup, move plate near sponge, move spoon near sponge, move orange cup near sponge, put coke can into sink, drop coke can into sink, push coke can into sink, put sponge into sink, drop sponge into sink, push sponge into sink, put green cloth into sink, drop green cloth into sink, push green cloth into sink
Unseen Environments (Hard)	pick coke can, pick apple, pick rxbar blueberry, move apple near coke can, move rxbar blueberry near apple, move coke can near rxbar blueberry, move coke can near stapler, move apple near stapler, move coke can near keyboard, move apple near keyboard, move coke can near tissue box, move apple near tissue box, move coke can near papers, move apple near papers, move coke can near mouse, move apple near mouse, move coke can near book, move apple near book, pick marker, pick stapler, pick mouse, move marker near apple, move stapler near apple, move mouse near apple, push coke can to the left, push coke can to the right, push sponge to the left, push sponge to the right, push tissue box to the left, push tissue box to the right, point at coke can, point at sponge, point at tissue box

Table 9: Natural-language instruction sets used in RT-2 [4] to evaluate generalization under distribution shifts, grouped by unseen objects, unseen visual backgrounds, and unseen environments, each with easy and hard variants.