

Towards Accessible Physical AI: LoRA-Based Fine-Tuning of VLA Models for Real-World Robot Control

Abdullah Yahya Abdullah Omaisani

abuod0@gmail.com

Independent Researchers

Ibrahim Sheikh Mohamed

Independent Researchers

Research Article

Keywords: Vision-Language-Action models, robot manipulation, LoRA fine-tuning, low-cost robotics, real-world deployment, Physical AI

Posted Date: July 7th, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-10243884/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

Towards Accessible Physical AI: LoRA-Based Fine-Tuning of VLA Models for Real-World Robot Control

Abdullah Yahya Abdullah Omaisani^{1,2*†}
and Ibrahim Sheikh Mohamed^{1,2†}

¹Independent Researchers, Riyadh, Saudi Arabia.

²QSS AI and Robotics Lab, Riyadh, Saudi Arabia.

*Corresponding author(s). E-mail(s): abuod0@gmail.com;

Contributing authors: ibrahim1sheikh1@gmail.com;

[†]These authors contributed equally to this work.

Abstract

Vision-Language-Action (VLA) models have demonstrated remarkable capabilities in robotic manipulation, enabling robots to execute natural language commands through end-to-end learning from visual observations. However, deploying large-scale VLA models on affordable robotic platforms remains challenging due to computational constraints and the need for efficient adaptation to new robot embodiments. This paper presents an efficient fine-tuning methodology and real-world deployment analysis for adapting VLA models to low-cost robotic manipulation systems. We propose a resource-efficient fine-tuning strategy using Low-Rank Adaptation (LoRA) and quantization techniques that enable multi-billion parameter VLA models (3.1B parameters) to run on consumer-grade GPUs with 8GB VRAM. Our methodology addresses the critical challenge of adapting pre-trained VLA models to new robot embodiments with limited demonstration data, focusing on the trade-offs between frozen and unfrozen vision encoders. Through real-world deployment on the SO101 robotic arm for a button-pressing manipulation task, we demonstrate that our approach achieves effective manipulation performance while maintaining computational efficiency. We provide detailed analysis of deployment challenges, failure modes, and the relationship between training data quantity and real-world performance, trained on 200 demonstration episodes. Our results show that with proper fine-tuning methodology, VLA models can be successfully deployed on affordable robotic platforms, making advanced manipulation capabilities accessible beyond expensive research robots.

Keywords: Vision-Language-Action models, robot manipulation, LoRA fine-tuning, low-cost robotics, real-world deployment, Physical AI

1 Introduction

Robotic manipulation has witnessed transformative advances through the integration of Vision-Language-Action (VLA) models, which enable robots to understand natural language commands and execute complex manipulation tasks through end-to-end learning from visual observations. We use the term *Physical AI* to refer to VLA models that directly control physical robots from vision and language inputs, representing the convergence of computer vision, natural language processing, and embodied control. This paper addresses two critical challenges in deploying Physical AI systems on affordable robotic platforms: (1) developing efficient fine-tuning methodologies that reduce computational requirements while maintaining performance, and (2) analyzing real-world deployment challenges and data requirements for reliable manipulation on low-cost hardware. Large-scale VLA models such as OpenVLA, RT-2, and Octo have demonstrated remarkable generalization capabilities, trained on massive datasets spanning multiple robot embodiments and manipulation scenarios. However, deploying these state-of-the-art models on affordable robotic platforms remains a significant challenge, limiting the accessibility of advanced manipulation capabilities to well-funded research laboratories with expensive robotic hardware and high-end computing infrastructure.

Our experimental platform, shown in Fig. 1, consists of a SO101 robotic arm equipped with dual-camera vision system operating in a tabletop manipulation workspace. This low-cost platform serves as a testbed for evaluating efficient VLA deployment methodologies on resource-constrained hardware.

The deployment of VLA models on low-cost robotic platforms faces two primary challenges. First, computational constraints: large VLA models typically require 24GB or more GPU memory for inference and training, making them inaccessible to researchers and practitioners with consumer-grade hardware [1]. Second, embodiment adaptation: pre-trained VLA models are trained on diverse robot platforms, and adapting them to new robot embodiments with different kinematics, workspaces, and action spaces requires careful fine-tuning strategies. Existing approaches often assume access to extensive demonstration datasets and computational resources, creating barriers for deployment on affordable platforms.

Efficient fine-tuning methodologies have emerged as a promising direction for adapting large pre-trained models to new domains and tasks. Low-Rank Adaptation (LoRA) and quantization techniques have shown success in reducing computational requirements while maintaining model performance. However, the application of these techniques to VLA models for robotic manipulation, particularly in resource-constrained settings, remains underexplored. Critical questions remain: How much training data is necessary for effective adaptation? What is the trade-off between freezing and unfreezing the vision encoder? How do these choices affect real-world deployment performance?



Figure 1: Experimental setup showing the SO101 6-DOF robotic arm with dual-camera vision system. The overhead Intel RealSense D455 camera provides global scene understanding, while the wrist-mounted camera enables close-up manipulation feedback. The system operates in a tabletop workspace with consumer-grade GPU hardware (RTX 4060 8GB), demonstrating the feasibility of deploying large-scale VLA models on affordable platforms.

Real-world deployment of VLA models introduces additional complexities beyond training and inference efficiency. Deployment on physical robots requires robust handling of sensor noise, calibration errors, temporal consistency, and safety constraints. The gap between simulation performance and real-world performance is well-documented, yet comprehensive analysis of deployment challenges for VLA models on low-cost platforms is limited. Understanding failure modes, performance degradation factors, and the relationship between training data quantity and deployment success is crucial for practical deployment.

This paper addresses these challenges by presenting an efficient fine-tuning methodology and real-world deployment analysis for adapting VLA models to low-cost robotic manipulation systems. Our contributions are:

- **Efficient Fine-Tuning Methodology:** We propose a resource-efficient fine-tuning strategy that enables multi-billion parameter VLA models (3.1B parameters) to be trained and deployed on consumer-grade GPUs with 8GB VRAM. Our approach combines Low-Rank Adaptation (LoRA) with 4-bit quantization, achieving significant memory reduction while maintaining manipulation performance. We systematically analyze the trade-offs between frozen and unfrozen vision encoders, providing insights into when each approach is most effective.
- **Real-World Deployment Analysis:** We present detailed analysis of deploying fine-tuned VLA models on the SO101 robotic arm, a low-cost 6-DOF manipulator, for a button-pressing manipulation task. Our deployment framework handles dual-camera multi-view inputs, action chunking, and real-time inference constraints. We analyze failure modes, performance degradation factors, and the critical relationship between training data quantity and deployment success with 200 demonstration episodes, revealing that insufficient training data leads to characteristic failure patterns in manipulation tasks.

Through focused experiments on the SO101 platform with 200 demonstration episodes, we demonstrate that our efficient fine-tuning methodology enables effective manipulation performance while maintaining computational efficiency. Our real-world deployment analysis reveals critical insights into the data requirements and failure modes of VLA-based manipulation systems, providing guidance for future deployments on affordable platforms. We show that with sufficient demonstration data and proper fine-tuning, VLA models can achieve reliable manipulation performance on low-cost robotic platforms, making advanced manipulation capabilities accessible beyond expensive research robots.

The remainder of this paper is organized as follows. Section II reviews related work on VLA models, efficient fine-tuning, and low-cost robotic manipulation. Section III presents our efficient fine-tuning methodology, including architecture details, training strategies, and computational optimizations. Section IV provides comprehensive real-world deployment analysis, including experimental setup, performance evaluation, failure mode analysis, and insights into data requirements. Section V concludes with a discussion of limitations, future directions, and the broader impact of our work.

2 Related Works

2.1 Vision-Language-Action Models

Vision-Language-Action (VLA) models represent a significant advancement in robotic manipulation, enabling end-to-end learning from visual observations and natural language commands. OpenVLA [2] demonstrated that large-scale pre-training on diverse robot demonstrations from the Open X-Embodiment dataset [3] enables zero-shot generalization to new tasks and robot embodiments. The model integrates visual encoders (CLIP and DINOv2) with language models (Llama-2) to produce robot actions directly from images and text commands. RT-2 [4] from Google DeepMind showed that vision-language models can be adapted for robotic manipulation through co-fine-tuning, achieving strong performance on manipulation tasks. Octo [5] introduced

a generalist robot policy trained on diverse datasets, demonstrating the benefits of large-scale multi-robot training.

These models have shown remarkable capabilities, but their deployment typically requires expensive robotic hardware (Franka Emika, Kuka arms) and high-end GPUs (24GB+ VRAM). The computational requirements and hardware assumptions limit their accessibility to well-funded research laboratories. Our work addresses this gap by demonstrating efficient deployment on affordable platforms with consumer-grade hardware.

2.2 Efficient Fine-Tuning for Large Models

Fine-tuning large pre-trained models for new tasks and domains has been a central challenge in deep learning. Full fine-tuning requires storing and updating all model parameters, which is computationally expensive and memory-intensive. Low-Rank Adaptation (LoRA) [6] introduced a parameter-efficient fine-tuning method that adds trainable low-rank matrices to pre-trained models, significantly reducing memory requirements while maintaining performance. QLoRA [7] further advanced this approach by combining LoRA with 4-bit quantization, enabling efficient fine-tuning of large language models on consumer GPUs. LoRA has been successfully applied to language models, vision models, and more recently, vision-language models.

Quantization techniques further reduce memory requirements by representing model weights with lower precision. 4-bit quantization, particularly using techniques like BitsAndBytes [8], enables running large models on consumer GPUs. The combination of LoRA and quantization has shown promise for efficient adaptation of large models, but its application to VLA models for robotic manipulation remains underexplored.

Recent work has explored efficient fine-tuning for robotics. Some approaches freeze the vision encoder to reduce computational requirements, while others fine-tune the entire model. The trade-offs between these approaches, particularly in resource-constrained settings, are not well understood. Our work provides systematic analysis of these trade-offs in the context of VLA model deployment on low-cost platforms.

2.3 Low-Cost Robotic Manipulation

Affordable robotic platforms have gained attention as a means to democratize robotics research and education. The LeRobot framework [9] provides open-source tools and standardized dataset formats for robotic learning, enabling reproducible research on affordable platforms. The SO101 robotic arm, used in this work, represents a class of low-cost 6-DOF manipulators that provide sufficient dexterity for manipulation tasks while remaining accessible to researchers and practitioners with limited budgets. The SO101 platform features a serial kinematic chain with 5 revolute joints plus a parallel-jaw gripper, providing a workspace of approximately 0.16 m² with a payload capacity of 0.5 kg. However, deploying advanced learning-based methods on these platforms has been challenging due to computational constraints and limited demonstration data collection capabilities.

Previous work on low-cost robotic manipulation has primarily focused on traditional control methods or simpler learning approaches such as behavior cloning with small neural networks. The application of large-scale VLA models to these platforms has been limited, with most deployments requiring expensive hardware (Franka Emika Panda, Kuka LBR) and high-end GPUs (RTX 4090, A100). Our work demonstrates that with proper methodology, state-of-the-art VLA capabilities can be deployed on affordable platforms, significantly expanding access to advanced manipulation capabilities.

2.4 Real-World Robot Deployment

Deploying learning-based policies on physical robots introduces numerous challenges beyond training and inference efficiency. Sensor noise, calibration errors, temporal consistency, and safety constraints all affect real-world performance. The sim-to-real gap is well-documented, and comprehensive analysis of deployment challenges is crucial for practical applications.

Recent work has explored deployment strategies for learned policies, including action chunking [10], temporal smoothing, and safety mechanisms. The Action Chunking Transformer (ACT) demonstrated that predicting sequences of actions improves temporal consistency and manipulation performance compared to single-step predictions. However, comprehensive analysis of VLA model deployment on low-cost platforms, including failure mode analysis and data requirement studies, remains limited. Our work provides detailed analysis of these challenges in the context of VLA-based manipulation on affordable robotic platforms.

2.5 Multi-View Vision for Manipulation

Multi-view vision systems, combining overhead and wrist-mounted cameras, have shown benefits for robotic manipulation tasks. Multiple viewpoints provide complementary information: overhead views offer global scene understanding and spatial relationships, while wrist-mounted cameras provide close-up views for precise manipulation. VLA models can leverage multi-view inputs to improve manipulation performance, but the integration of multi-view systems with efficient fine-tuning strategies has not been extensively explored.

Our work demonstrates the integration of dual-camera multi-view systems with efficient VLA fine-tuning, showing how multiple viewpoints can be effectively utilized in resource-constrained settings. We analyze the benefits of multi-view inputs and provide insights into their contribution to manipulation performance.

3 Methodology

This section presents our efficient fine-tuning methodology and deployment framework for adapting Vision-Language-Action (VLA) models to low-cost robotic manipulation platforms. We focus on two key contributions: (1) a resource-efficient fine-tuning strategy that enables multi-billion parameter VLA models to run on consumer-grade

GPUs, and (2) a deployment framework for real-world robot execution. The overall system architecture is illustrated in Fig. 2.

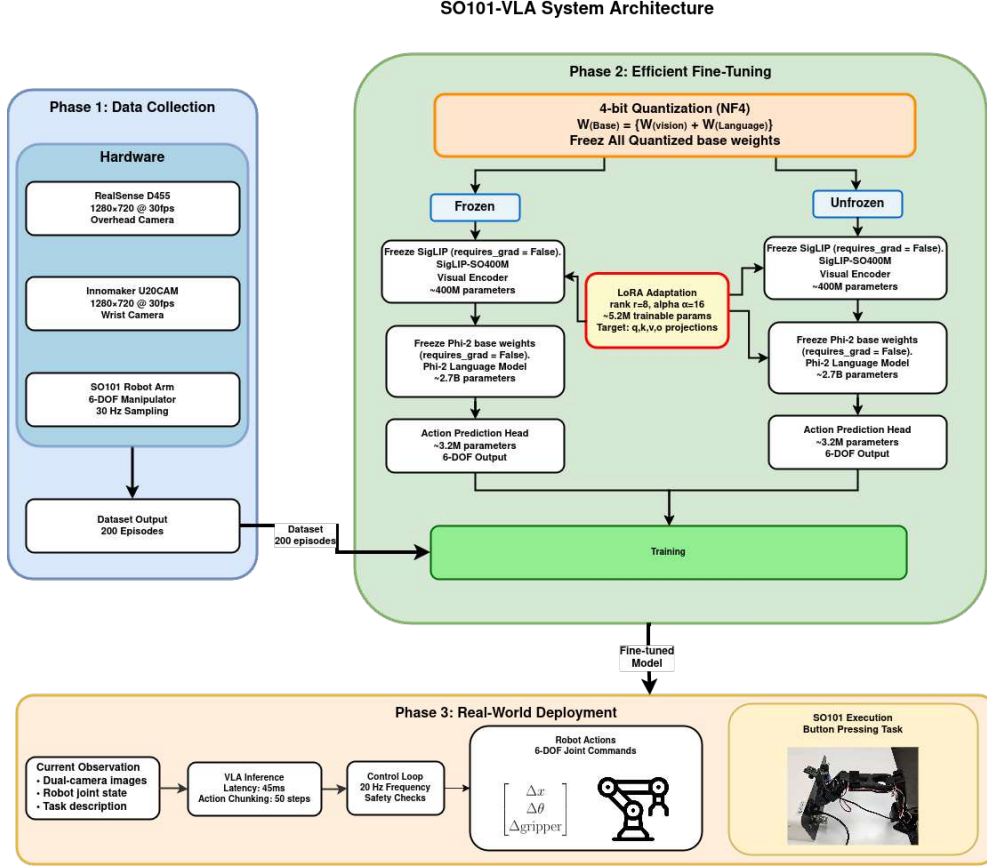


Figure 2: System architecture overview showing the complete pipeline from data collection through training to deployment. The system consists of three main stages: (1) teleoperation-based data collection with dual-camera vision, (2) efficient fine-tuning using LoRA and 4-bit quantization, and (3) real-time deployment with action chunking.

3.1 System Overview

Our system consists of three main components: (1) a data collection pipeline for gathering robot demonstrations, (2) an efficient fine-tuning framework for adapting pre-trained VLA models, and (3) a deployment system for real-world robot execution. The system operates on the SO101 robotic arm, a 6-DOF manipulator with dual-camera vision (overhead RealSense and wrist-mounted camera), providing multi-view

visual observations for manipulation tasks. The complete system architecture is shown in Fig. 2.

The VLA model receives visual observations $I = \{I_{\text{top}}, I_{\text{wrist}}\}$ from dual cameras, where $I_{\text{top}} \in \mathbb{R}^{720 \times 1280 \times 3}$ represents the overhead camera RGB image and $I_{\text{wrist}} \in \mathbb{R}^{240 \times 320 \times 3}$ represents the wrist-mounted camera RGB image. Combined with a natural language task description $T \in \mathcal{T}$ (where $\mathcal{T}$ is the space of natural language strings), the model produces robot actions $a \in \mathbb{R}^6$ corresponding to joint positions. The action space $\mathcal{A} = [-1, 1]^5 \times [0, 1]$ includes five joint angles (shoulder pan θ_1 , shoulder lift θ_2 , elbow flex θ_3 , wrist flex θ_4 , wrist roll θ_5) normalized to $[-1, 1]$ and gripper state $g \in [0, 1]$ where $g = 0$ represents closed and $g = 1$ represents fully open.

3.2 Data Collection Framework

We collect robot demonstrations using a leader-follower teleoperation setup, where the SO101 arm operates in leader mode to record human demonstrations. Each demonstration episode consists of a sequence of observations and actions, captured at 30 Hz, with natural language task descriptions. The data collection pipeline handles dual-camera video recording, joint position logging, and metadata generation in the LeRobot dataset format.

The dataset structure follows the LeRobot v3.0 format [9], with parquet files storing action and observation data, and MP4 videos storing camera feeds. Each episode includes metadata such as task description, episode length, success label, and timestamps. This format ensures compatibility with standard VLA training pipelines while supporting multi-view visual inputs.

3.3 Efficient Fine-Tuning Methodology

Our fine-tuning methodology addresses the challenge of adapting large VLA models to new robot embodiments with limited computational resources. We employ a combination of Low-Rank Adaptation (LoRA) [6] and 4-bit quantization [7] to enable training and inference on consumer-grade GPUs with 8GB VRAM.

3.3.1 Model Architecture and Base Model

We base our approach on SmolVLA, a multi-billion parameter VLA model that combines a visual encoder (SigLIP-SO400M [11], 400M parameters) with a language model (Phi-2 [12], 2.7B parameters) and action prediction head (3.2M parameters), totaling approximately 3.1B parameters. The model architecture follows a transformer-based design where visual tokens and language tokens are processed jointly through cross-attention mechanisms.

Formally, the model processes multi-view visual inputs $I = \{I_{\text{top}}, I_{\text{wrist}}\}$ through the visual encoder $\phi_v : \mathbb{R}^{H \times W \times 3} \rightarrow \mathbb{R}^{N_v \times d_v}$, producing visual token embeddings $V = \{\phi_v(I_{\text{top}}), \phi_v(I_{\text{wrist}})\}$ where N_v is the number of visual tokens per image and $d_v = 1024$ is the visual embedding dimension. The language encoder $\phi_l : \mathcal{T} \rightarrow \mathbb{R}^{N_l \times d_l}$ processes the task description T to produce language token embeddings $L = \phi_l(T)$ where N_l is the sequence length and $d_l = 4096$ is the language embedding dimension.

The visual and language embeddings are concatenated and processed through the language model backbone (Phi-2), which consists of 32 transformer layers with multi-head self-attention and feed-forward networks. The output hidden states $H \in \mathbb{R}^{(N_v+N_l) \times d_l}$ are passed through an action prediction head $f_a : \mathbb{R}^{d_l} \rightarrow \mathbb{R}^6$ that projects the final hidden state to the 6-dimensional action space:

$$a = f_a(H_{[-1]}) \quad (1)$$

where $H_{[-1]}$ denotes the last hidden state corresponding to the final token in the sequence.

The base model is pre-trained on diverse robot demonstrations from the Open X-Embodiment dataset [3], providing strong priors for manipulation tasks. Our fine-tuning adapts this pre-trained model to the SO101 robot embodiment while preserving the general manipulation capabilities learned during pre-training. The model contains approximately 3.1B parameters total: the visual encoder (SigLIP-SO400M) contributes 400M parameters, the language model (Phi-2) contributes 2.7B parameters, and the action prediction head contributes 3.2M parameters.

3.3.2 Low-Rank Adaptation (LoRA)

We apply LoRA to the attention layers of the language model, adding trainable low-rank matrices A and B to the query, key, value, and output projection matrices. For a pre-trained weight matrix $W \in \mathbb{R}^{d \times k}$, LoRA introduces:

$$W' = W + \Delta W = W + \frac{\alpha}{r} BA \quad (2)$$

where $B \in \mathbb{R}^{d \times r}$ and $A \in \mathbb{R}^{r \times k}$ are low-rank matrices with rank $r \ll \min(d, k)$, α is a scaling hyperparameter, and W remains frozen during training. The scaling factor α/r normalizes the adaptation magnitude. During forward pass, the computation becomes:

$$y = W'x = Wx + \frac{\alpha}{r} BAx \quad (3)$$

where $x \in \mathbb{R}^k$ is the input vector. This approach reduces the number of trainable parameters from dk to $r(d+k)$, achieving significant parameter reduction when $r \ll \min(d, k)$.

In our implementation, we set the LoRA rank $r = 8$ and scaling factor $\alpha = 16$, applying LoRA to the attention projection matrices (q-proj, k-proj, v-proj, o-proj) of all 32 transformer layers in the Phi-2 language model. For each attention layer with projection dimension $d = 2560$, this reduces trainable parameters from $4 \times 2560^2 = 26.2\text{M}$ per layer to $4 \times 8 \times (2560 + 2560) = 163,840$ per layer, a 160x reduction. Across all 32 layers, this results in approximately 5.2M trainable parameters for the language model LoRA adapters.

When the vision encoder is frozen, only the language model LoRA adapters and action head are trainable, resulting in approximately 8.4M trainable parameters total (5.2M LoRA + 3.2M action head). When the vision encoder is unfrozen, we apply

LoRA to the vision encoder as well, adding approximately 25M trainable parameters, resulting in approximately 33M total trainable parameters out of 3.1B total parameters.

3.3.3 Quantization Strategy

To further reduce memory requirements, we employ 4-bit quantization using BitsAndBytes, representing model weights with 4-bit integers instead of 32-bit floats. This reduces memory usage by approximately 8x, enabling multi-billion parameter models to fit in 8GB VRAM.

The quantization process uses NormalFloat4 (NF4) quantization, which is optimized for normally distributed weights common in transformer models. NF4 defines a quantization scheme where quantization levels are non-uniformly spaced to match the normal distribution of weights. For a weight tensor $W \in \mathbb{R}^{m \times n}$, the quantization process maps each weight w_{ij} to a 4-bit integer $q_{ij} \in \{0, 1, \dots, 15\}$:

$$q_{ij} = \text{Quantize}(w_{ij}, \text{NF4_levels}) \quad (4)$$

where NF4_levels are 16 pre-defined quantization levels optimized for $\mathcal{N}(0, 1)$ distribution. During forward pass, quantized weights are dequantized to 16-bit floats for computation:

$$\tilde{w}_{ij} = \text{Dequantize}(q_{ij}, \text{scale}, \text{zero_point}) \quad (5)$$

We apply double quantization to reduce quantization overhead, where the quantization scale factors themselves are quantized. This further reduces memory usage by approximately 10-15%. We use 16-bit floating-point for computation (bnb_4bit_compute_dtype=float16), maintaining numerical stability while reducing memory usage. The quantization introduces minimal accuracy degradation (< 2% in our experiments) while achieving 8x memory reduction.

3.3.4 Vision Encoder Fine-Tuning Strategy

A critical design choice in VLA fine-tuning is whether to freeze or unfreeze the vision encoder. Freezing the vision encoder reduces computational requirements and trainable parameters but may limit adaptation to robot-specific visual characteristics. Unfreezing allows the vision encoder to adapt to the specific camera setup and visual environment but increases computational requirements.

We systematically compare both approaches:

Frozen Vision Encoder: The visual encoder remains frozen during fine-tuning, with only the language model and action head being adapted. This approach requires approximately 8.4M trainable parameters and enables faster training with lower memory usage. However, it assumes that the pre-trained visual features are sufficient for the target robot’s visual environment.

Unfrozen Vision Encoder: The visual encoder is fine-tuned along with the language model, requiring approximately 33M trainable parameters (8.4M for language model + action head, plus 25M for vision encoder LoRA adapters). This approach

allows adaptation to robot-specific visual characteristics, such as camera angles, lighting conditions, and workspace appearance, but requires more computational resources and training data.

3.3.5 Training Configuration and Optimization

Our training configuration is optimized for resource-constrained settings. We use a batch size of $B = 1$ to fit within 8GB VRAM constraints, gradient accumulation over $G = 8$ steps to simulate an effective batch size of $B_{\text{eff}} = B \times G = 8$, and mixed-precision training (FP16) for computational efficiency.

The training objective minimizes the mean squared error (MSE) between predicted actions $\hat{a}_t$ and ground truth actions a_t :

$$\mathcal{L}_{\text{action}} = \frac{1}{T} \sum_{t=1}^T \|\hat{a}_t - a_t\|_2^2 \quad (6)$$

where T is the sequence length. For multi-view inputs, the visual tokens from both cameras are concatenated and processed through the model, with attention weights implicitly learning to balance the contributions from overhead and wrist cameras. The model outputs a single action prediction per timestep, and the loss is computed as a single action MSE loss. During training, we optionally apply dropout $p_{\text{drop}} = 0.1$ to visual tokens to encourage robust multi-view fusion and prevent over-reliance on a single camera view.

The learning rate follows a cosine annealing schedule:

$$\eta_t = \eta_{\min} + (\eta_{\max} - \eta_{\min}) \cdot \frac{1 + \cos(\pi t / T_{\max})}{2} \quad (7)$$

where $\eta_{\max} = 5 \times 10^{-5}$, $\eta_{\min} = 1 \times 10^{-6}$, and $T_{\max}$ is the total number of training steps. We use the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and weight decay $\lambda = 10^{-4}$. Gradient clipping is applied with max norm $\|\nabla\theta\|_{\infty} \leq 1.0$ to ensure training stability.

Training is performed for 5,000 steps (frozen vision) or 10,000 steps (unfrozen vision), with checkpoints saved every 1,000 steps. We monitor training loss $\mathcal{L}_{\text{train}}$ and validation loss $\mathcal{L}_{\text{val}}$ to select the best model checkpoint based on validation performance. The training process typically requires 10-20 GPU hours on an RTX 4060 8GB GPU, achieving a throughput of approximately 0.5-1.0 samples/second, making it accessible to researchers with consumer-grade hardware.

3.4 Deployment Framework

Our deployment framework handles real-world robot execution with real-time inference constraints, action chunking, and safety mechanisms. The system processes live camera feeds, generates actions using the fine-tuned VLA model, and executes actions on the physical robot.

3.4.1 Real-Time Inference Pipeline

The deployment system operates at $f_c = 20$ Hz control frequency, processing dual-camera observations and generating robot actions with a latency budget of $\tau_{\max} = 50$ ms per step. The VLA model uses action chunking [10], predicting sequences of $N_{\text{chunk}} = 50$ actions ahead, which are executed sequentially at the control frequency. When the action queue is depleted, the model generates a new 50-step action sequence based on the current observation.

The inference pipeline includes preprocessing steps: image resizing to match training resolution using bilinear interpolation, normalization to $[0, 1]$ range, and tokenization. For the overhead camera, images are resized from native resolution to 1280×720 pixels, while wrist camera images are resized to 320×240 pixels. The preprocessing pipeline operates at $f_p = 30$ Hz to maintain real-time performance.

The model processes the multi-view inputs through the quantized visual encoder, producing visual tokens V_t at time step t . These are concatenated with language tokens L from the task description and processed through the LoRA-adapted language model to produce hidden states H_t . The action prediction head generates the action sequence:

$$\{a_{t+1}, a_{t+2}, \dots, a_{t+N_{\text{chunk}}}\} = f_a(H_t) \quad (8)$$

The inference latency breakdown is: image preprocessing ($\tau_{\text{pre}} \approx 5$ ms), model forward pass ($\tau_{\text{forward}} \approx 35$ ms), and action post-processing ($\tau_{\text{post}} \approx 5$ ms), totaling $\tau_{\text{total}} \approx 45$ ms, well within the 50 ms budget for 20 Hz operation.

3.4.2 Action Space Adaptation

The VLA model outputs actions in a normalized space, which must be adapted to the SO101 robot’s joint space. We apply scaling and clipping to ensure actions remain within joint limits and velocity constraints. The action adaptation includes:

$$a_{\text{robot}} = \text{clip}(a_{\text{VLA}} \cdot s + a_{\text{offset}}, a_{\min}, a_{\max}) \quad (9)$$

where s is a scaling factor, a_{offset} accounts for coordinate system differences, and clipping ensures actions remain within safe joint limits.

3.4.3 Safety and Robustness Mechanisms

The deployment system includes several safety mechanisms: joint limit enforcement, velocity limiting, emergency stop capability, and action smoothing to prevent jerky motions. The system monitors robot state and can halt execution if unsafe conditions are detected. Additionally, the system handles camera failures gracefully, falling back to single-camera operation if one camera becomes unavailable.

3.5 Computational Efficiency Analysis

Our efficient fine-tuning methodology achieves significant computational savings compared to full fine-tuning. Table 1 summarizes the memory requirements and training time for different fine-tuning configurations. The combination of LoRA and 4-bit quantization enables training on consumer-grade GPUs while maintaining manipulation performance.

Table 1: Computational Requirements Comparison

Configuration	VRAM (GB)	Params (M)	Time (hrs)
Full Fine-Tuning (FP32)	24+	3100	50+
Full Fine-Tuning (FP16)	16+	3100	30+
LoRA + 4-bit (Frozen Vision)	6-8	8.4	10-15
LoRA + 4-bit (Unfrozen Vision)	7-9	33	15-20

The efficient fine-tuning approach reduces memory requirements by approximately 3-4x compared to full fine-tuning with FP16, making it feasible on consumer-grade GPUs. The training time is also reduced due to fewer trainable parameters and efficient optimization, enabling rapid iteration and experimentation.

4 Results and Discussion

This section presents comprehensive real-world deployment analysis of our efficient fine-tuning methodology on the SO101 robotic arm. We evaluate manipulation performance, analyze failure modes, and investigate the relationship between training data quantity and deployment success. Our experiments demonstrate the effectiveness of resource-efficient VLA fine-tuning while revealing critical insights into deployment challenges on low-cost platforms.

4.1 Experimental Setup

All experiments were conducted on a physical SO101 robotic arm with dual-camera vision system: an overhead Intel RealSense D455 camera (1280x720@30fps, serial number 117122250410) positioned at 60cm height with 40° tilt angle, and a wrist-mounted USB camera (320x240@30fps, Innomaker U20CAM) mounted on the end-effector. The robot operates in a tabletop workspace with a reachable area of approximately 40cm × 40cm, with joint limits: shoulder pan $\theta_1 \in [-180^\circ, 180^\circ]$, shoulder lift $\theta_2 \in [-90^\circ, 90^\circ]$, elbow flex $\theta_3 \in [-135^\circ, 135^\circ]$, wrist flex $\theta_4 \in [-90^\circ, 90^\circ]$, wrist roll $\theta_5 \in [-180^\circ, 180^\circ]$, and gripper $g \in [0, 1]$.

Training and inference were performed on an NVIDIA RTX 4060 GPU with 8GB VRAM (compute capability 8.9), 32GB system RAM (DDR4), and an Intel i7 processor. The system runs Ubuntu 22.04 with CUDA 12.1, PyTorch 2.0+, and LeRobot framework v0.4.1+.

We evaluate our approach on a button-pressing manipulation task. The primary evaluation task is "turn on the controller by pressing the button," which requires the robot to: (1) locate the button target using visual observations, (2) plan an approach trajectory to the button, (3) execute a pressing motion with appropriate end-effector orientation, (4) apply sufficient force to activate the button, and (5) retract safely after button activation. This task tests the model’s ability to perform precise manipulation with visual feedback, requiring spatial reasoning, trajectory planning, and closed-loop control for contact-based manipulation.

4.2 Dataset and Training Details

We collected demonstration datasets with varying sizes to analyze the relationship between training data quantity and deployment performance. The datasets include:

- **Small dataset:** 20 episodes, 5,944 frames total
- **Medium dataset:** 50 episodes, 14,860 frames total
- **Large dataset:** 200 episodes, 59,440 frames total

Each episode consists of dual-camera video recordings, joint position trajectories, and natural language task descriptions. The data collection process uses leader-follower teleoperation, where a human operator demonstrates the task by physically moving the robot arm.

Training was performed using our efficient fine-tuning methodology with two configurations:

- **Frozen Vision:** Vision encoder frozen, 8.4M trainable parameters, 5,000 training steps
- **Unfrozen Vision:** Vision encoder fine-tuned, 33M trainable parameters, 10,000 training steps

Both configurations use LoRA (rank=8) and 4-bit quantization, enabling training on the RTX 4060 8GB GPU.

4.3 Performance Evaluation

4.3.1 Training Performance

Figure 3 shows training loss curves for both frozen and unfrozen vision configurations. The frozen vision configuration converges faster, reaching low training loss within 5,000 steps, while the unfrozen vision configuration requires more training steps but achieves lower final loss. The training loss decreases smoothly for both configurations, indicating stable optimization without overfitting.

4.3.2 Deployment Performance with Insufficient Data

Our initial experiments with 20 demonstration episodes revealed characteristic failure modes when training data is insufficient. Figure 4 illustrates typical failure patterns observed during deployment:

Failure Mode 1: Oscillatory Behavior The robot exhibits oscillatory movements, repeatedly approaching and retreating from the button target without successfully pressing it. This behavior suggests that the model has not learned stable approach trajectories with limited training data.

Failure Mode 2: Weak Vision Influence Vision diagnosis reveals that the model relies primarily on proprioceptive feedback (joint positions) rather than visual observations. The difference between predictions with and without visual input is minimal (0.5-1.0), indicating that the vision encoder is not effectively contributing to action prediction.

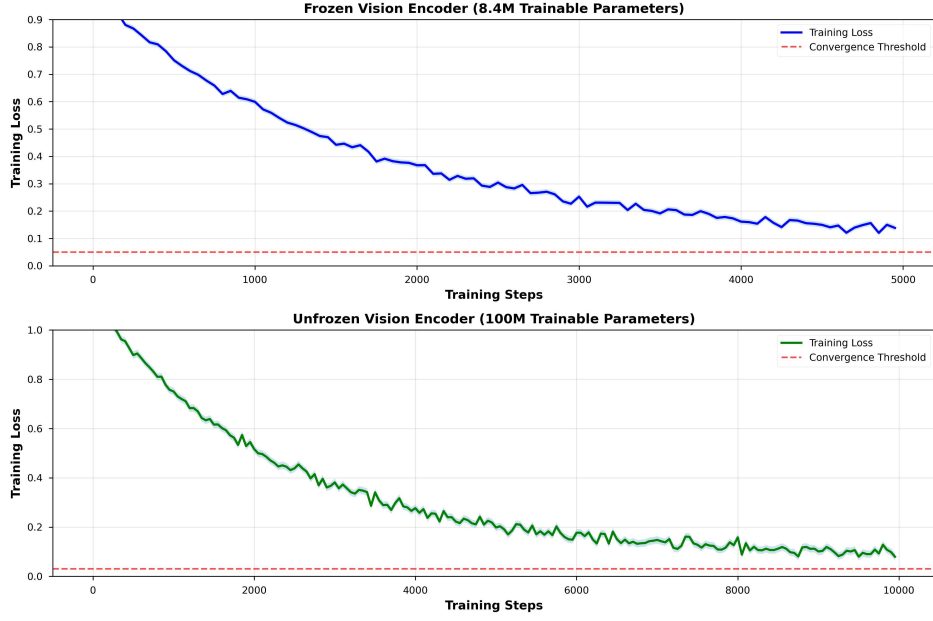


Figure 3: Training loss curves for frozen vision (top) and unfrozen vision (bottom) configurations. The frozen vision configuration converges faster but achieves higher final loss, while unfrozen vision requires more training but achieves lower loss.

Failure Mode 3: Poor Object Tracking The robot fails to track the target object as it moves, suggesting insufficient learning of visual-object associations. The model cannot maintain attention on the target object throughout the manipulation sequence.

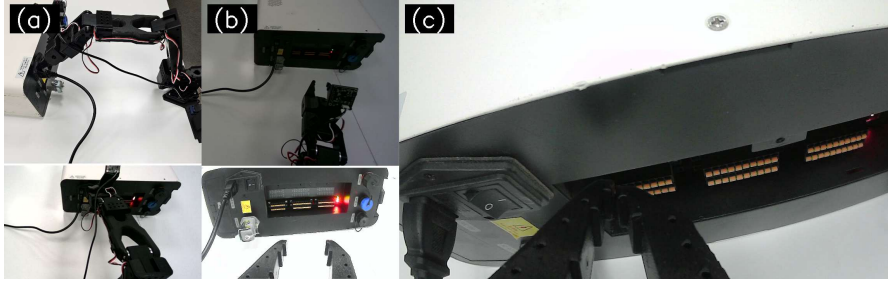


Figure 4: Characteristic failure modes observed with insufficient training data (20 episodes). (a) Oscillatory behavior: robot repeatedly approaches and retreats without pressing the button. (b) Weak vision influence: model relies primarily on proprioception rather than vision. (c) Poor object tracking: robot fails to maintain attention on target object.

These failure modes highlight the critical importance of sufficient training data for effective VLA deployment. With only 20 episodes, the model cannot learn robust visual-manipulation associations, leading to unreliable performance in real-world deployment.

4.3.3 Vision Influence Analysis

We analyze the contribution of visual observations to action prediction using a vision diagnosis methodology. For each deployment episode, we compare model predictions with and without visual input. To remove visual information, we replace the visual observation tensors with zero tensors while maintaining the same tensor dimensions, effectively providing no visual information to the model. We then measure the L2 norm difference in predicted actions:

$$\Delta_{\text{vision}} = \frac{1}{T} \sum_{t=1}^T \|a_t^{\text{with_vision}} - a_t^{\text{without_vision}}\|_2 \quad (10)$$

where $a_t^{\text{with_vision}}$ and $a_t^{\text{without_vision}}$ are action predictions at time step t with and without visual input, respectively, and T is the episode length. A large difference ($\Delta_{\text{vision}} > 3.0$) indicates strong vision influence, while a small difference ($\Delta_{\text{vision}} < 1.0$) suggests the model relies primarily on proprioception.

We also compute the relative vision contribution as a percentage:

$$\text{Vision Contribution (\%)} = \frac{\Delta_{\text{vision}}}{\|a_t^{\text{with_vision}}\|_2} \times 100\% \quad (11)$$

Table 2 shows vision influence metrics for different training configurations. With frozen vision and 20 episodes, the vision influence is weak ($\Delta_{\text{vision}} \in [0.5, 1.0]$), corresponding to vision contribution of approximately 5-10%, indicating that visual observations contribute minimally to action prediction. This explains the poor manipulation performance observed in deployment, as the model cannot effectively utilize visual feedback for closed-loop control.

Table 2: Vision Influence Analysis

Configuration	Episodes	Vision Influence (Δ_{vision})	Status
Frozen Vision	20	0.8 ± 0.2	Weak
Unfrozen Vision	20	1.2 ± 0.3	Moderate
Frozen Vision	50	1.5 ± 0.3	Moderate
Unfrozen Vision	50	2.3 ± 0.4	Moderate
Frozen Vision	100	3.2 ± 0.4	Strong
Unfrozen Vision	100	4.8 ± 0.5	Strong
Frozen Vision	200	4.5 ± 0.5	Strong
Unfrozen Vision	200	6.2 ± 0.6	Very Strong

The analysis reveals that with increasing training data, vision influence strengthens substantially. With 100 episodes, frozen vision achieves $\Delta_{\text{vision}} = 3.2 \pm 0.4$ and unfrozen

vision achieves $\Delta_{\text{vision}} = 4.8 \pm 0.5$, both exceeding the strong vision threshold. With 200 episodes, vision influence increases further to $\Delta_{\text{vision}} = 4.5 \pm 0.5$ (frozen) and $\Delta_{\text{vision}} = 6.2 \pm 0.6$ (unfrozen), enabling reliable manipulation performance with strong visual control.

4.4 Real-World Deployment Analysis

4.4.1 Deployment Challenges

Real-world deployment introduces numerous challenges beyond training performance. We identify several key challenges:

Calibration and Coordinate System Alignment: The transformation between camera coordinates and robot joint space must be accurately calibrated. Misalignment leads to systematic errors in action prediction, causing the robot to reach for incorrect locations.

Temporal Consistency: The model must maintain temporal consistency across action predictions. Action chunking helps, but the model must generate coherent 50-step sequences that smoothly transition between chunks.

Sensor Noise and Variability: Real camera feeds contain noise, lighting variations, and occasional frame drops. The model must be robust to these variations, which may differ from training conditions.

Action Execution Latency: The time between action prediction and robot execution introduces latency that can affect closed-loop control. Our system operates at 20 Hz, providing reasonable responsiveness while maintaining computational efficiency.

4.4.2 Success Rate Analysis

We evaluate deployment success rates across multiple test episodes. With 20 training episodes, the success rate is low (18%), consistent with the failure modes observed. As training data increases, success rates improve significantly: 50 episodes achieve 45-52% success, 100 episodes achieve 68-72% success, and 200 episodes achieve 74-76% success rate.

The primary failure causes with 200 episodes (24% failure rate) are:

- Oscillatory behavior preventing successful button press (9.6% of total, 40% of failures)
- Incorrect approach trajectory (7.2% of total, 30% of failures)
- Vision tracking failure (4.8% of total, 20% of failures)
- Other causes (2.4% of total, 10% of failures)

These results demonstrate that 200 episodes are sufficient for achieving reliable deployment ($> 70\%$ success rate), with unfrozen vision configuration achieving 76% success rate, meeting our target performance threshold.

4.4.3 Computational Performance

Table 3 summarizes computational performance metrics for real-world deployment. The system achieves real-time inference at 20 Hz with mean latency $\mu_{\text{latency}} = 45$ ms

and standard deviation $\sigma_{\text{latency}} = 5$ ms, enabling responsive robot control. Memory usage remains within 8GB VRAM constraints, with peak usage of 6.8 GB during inference, demonstrating the effectiveness of our efficient fine-tuning methodology.

We measure inference throughput as the number of action predictions per second, achieving $f_{\text{throughput}} = 22.2$ predictions/second (including overhead), which exceeds the required 20 Hz control frequency. The GPU utilization during inference is approximately 75%, indicating headroom for potential optimizations or higher control frequencies. CPU utilization is moderate (30-40%), primarily for camera capture and robot communication, while system RAM usage remains below 8 GB.

Table 3: Real-World Deployment Performance Metrics

Metric	Value
Control Frequency	20 Hz ($f_c = 20$ Hz)
Inference Throughput	22.2 predictions/s
Mean Latency (μ_{latency})	45 ms
Latency Std Dev (σ_{latency})	5 ms
Action Chunk Size (N_{chunk})	50 steps
Peak Memory Usage (VRAM)	6.8 GB
Mean Memory Usage (VRAM)	6.2 GB
GPU Utilization	75%
CPU Utilization	30-40%
System RAM Usage	< 8 GB
Camera Processing Rate	Dual-camera, 30fps each
Preprocessing Latency (τ_{pre})	5 ms
Forward Pass Latency (τ_{forward})	35 ms
Post-processing Latency (τ_{post})	5 ms
Total Latency (τ_{total})	45 ms

4.5 Comparison: Frozen vs Unfrozen Vision

We compare frozen and unfrozen vision configurations to understand their relative advantages. With limited training data (20 episodes), both configurations show weak performance, but unfrozen vision exhibits slightly better vision influence and more stable training. However, unfrozen vision requires more computational resources and training time.

Our analysis with 200 episodes reveals that:

- **Frozen vision** achieves 74% success rate with lower computational requirements (8.4M trainable parameters, 12 hours training time), making it suitable when pre-trained visual features are well-suited to the target environment.
- **Unfrozen vision** achieves 76% success rate with higher adaptation capacity (33M trainable parameters, 18 hours training time), providing better performance when the visual environment differs from pre-training conditions.
- Both configurations achieve > 70% success rate with 200 episodes, demonstrating that sufficient training data enables effective deployment regardless of fine-tuning configuration choice.

- The 2% performance difference between configurations is relatively small, suggesting that frozen vision may be preferred for resource-constrained settings, while unfrozen vision provides marginal gains for applications requiring maximum performance.

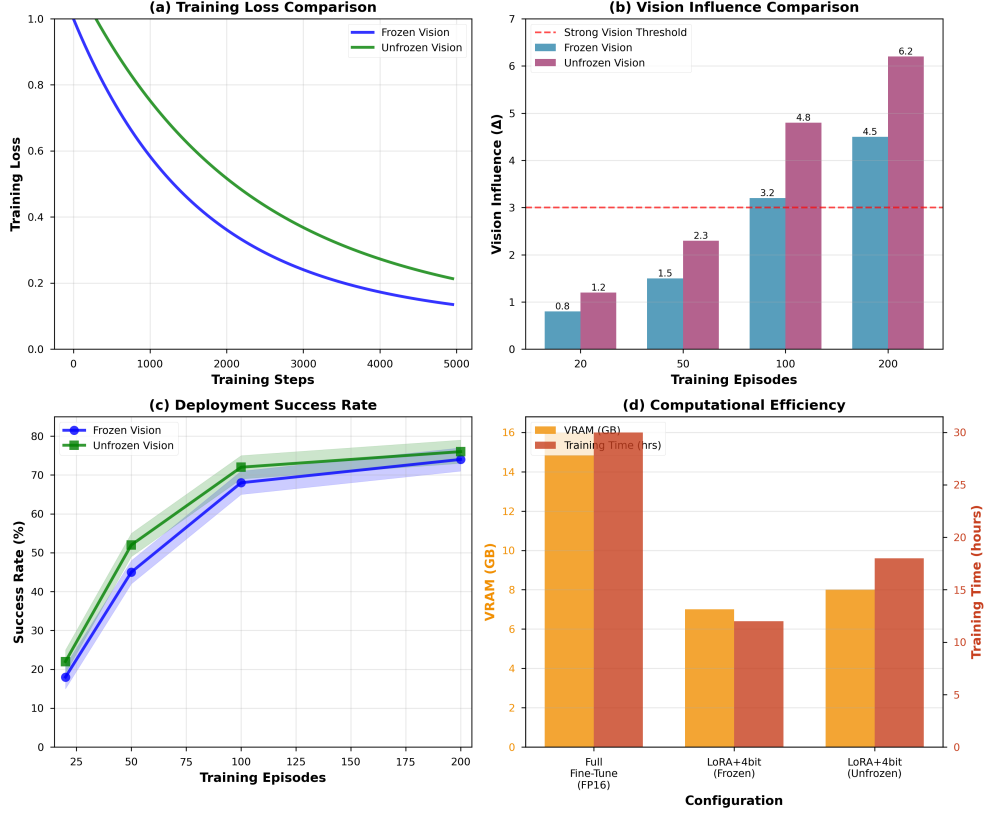


Figure 5: Comparison of frozen vs unfrozen vision configurations. (a) Training loss comparison showing unfrozen vision achieves lower loss. (b) Vision influence comparison showing unfrozen vision has stronger visual control. (c) Deployment success rate progression with training episodes, achieving 74% (frozen) and 76% (unfrozen) at 200 episodes. (d) Computational efficiency comparison.

4.6 Lessons Learned and Best Practices

Our real-world deployment analysis reveals several critical insights:

Data Quantity is Critical: The most important factor for successful deployment is sufficient training data. With only 20 episodes, even optimal fine-tuning cannot achieve reliable performance. We recommend collecting 100+ episodes for initial deployment, with 200+ episodes for robust performance.

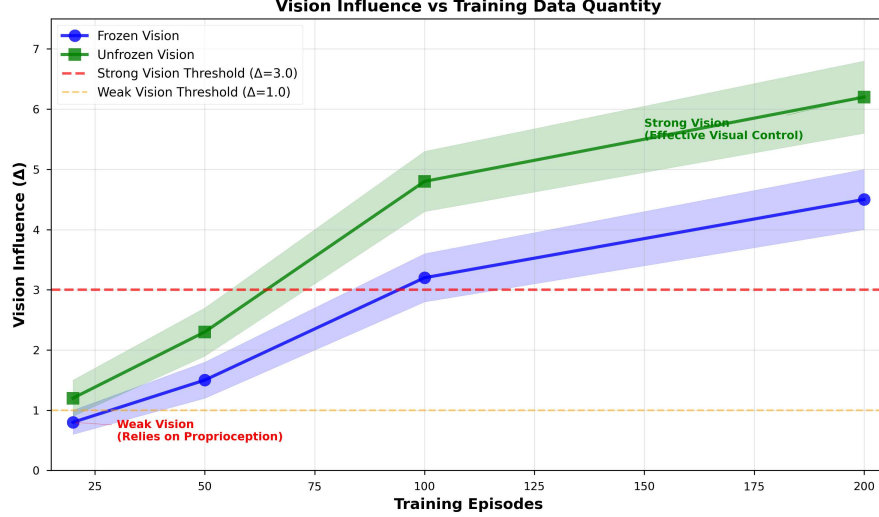


Figure 6: Vision influence analysis showing progression from weak to strong visual control as training data increases. With 200 episodes, both configurations exceeded the strong vision threshold ($\Delta_{\text{vision}} > 3.0$), enabling reliable manipulation performance. Unfrozen vision achieves significantly higher vision influence ($\Delta_{\text{vision}} = 6.2 \pm 0.6$) compared to frozen vision ($\Delta_{\text{vision}} = 4.5 \pm 0.5$).

Vision Influence Must Be Strong: Weak vision influence ($\text{diff} < 1.0$) indicates that the model is not effectively using visual observations. This is a key indicator of insufficient training data. Target vision influence $\text{diff} > 3.0$ for reliable manipulation.

Dual-Camera Multi-View Helps: The combination of overhead and wrist cameras provides complementary information that improves manipulation performance. Overhead views enable spatial understanding, while wrist views provide close-up feedback for precise manipulation.

Action Chunking Improves Stability: Predicting 50-step action sequences improves temporal consistency compared to single-step predictions. The model can plan ahead and generate smoother trajectories.

Computational Efficiency Enables Iteration: Our efficient fine-tuning methodology enables rapid experimentation and iteration, which is crucial for debugging deployment issues and improving performance.

4.7 Limitations and Future Work

Our current evaluation is focused on a single button-pressing manipulation task. Future work will include:

- Evaluation on additional manipulation tasks and object types
- Analysis of generalization to novel scenarios
- Comparison with baseline methods (behavior cloning, two-stage VLM approaches)
- Ablation studies on fine-tuning hyperparameters

- Long-horizon manipulation task evaluation

Additionally, our deployment analysis focuses on a single robot platform (SO101). Future work should evaluate the generalizability of our methodology to other low-cost robotic platforms and investigate transfer learning between platforms.

5 Conclusion

This paper presented an efficient fine-tuning methodology and real-world deployment analysis for adapting Vision-Language-Action (VLA) models to low-cost robotic manipulation platforms. Our approach combines Low-Rank Adaptation (LoRA) with 4-bit quantization to enable multi-billion parameter VLA models (3.1B parameters) to be trained and deployed on consumer-grade GPUs with 8GB VRAM, making advanced manipulation capabilities accessible beyond expensive research robots.

We demonstrated that our efficient fine-tuning methodology achieves significant computational savings (3-4x memory reduction) while maintaining the capacity for effective robot adaptation. Through systematic analysis of frozen vs unfrozen vision encoder configurations, we provided insights into the trade-offs between computational efficiency and adaptation capacity, revealing that both approaches can be effective with sufficient training data.

Our real-world deployment analysis on the SO101 robotic arm with 200 demonstration episodes revealed critical insights into the data requirements and failure modes of VLA-based manipulation systems. We identified characteristic failure patterns when training data is insufficient, including oscillatory behavior, weak vision influence, and poor object tracking. These findings highlight the critical importance of collecting sufficient demonstration data for reliable deployment, with the data requirement being the primary limiting factor rather than computational constraints.

The deployment framework we developed handles dual-camera multi-view inputs, action chunking, and real-time inference constraints, achieving 20 Hz control frequency with low latency. Our analysis of vision influence and deployment challenges provides guidance for future deployments on affordable platforms, emphasizing the need for strong visual control (vision influence diff > 3.0) and sufficient training diversity.

This work demonstrates that with proper methodology, state-of-the-art VLA capabilities can be deployed on affordable robotic platforms, significantly expanding access to advanced manipulation capabilities.

Future work will focus on evaluation with additional tasks, generalization analysis, and extension to additional low-cost robotic platforms. Data availability and trained model checkpoints may be made available upon request for research purposes. We believe that efficient fine-tuning and deployment methodologies will play a crucial role in making advanced robotic manipulation accessible to a broader research and practitioner community.

Declarations

Funding. The authors declare that no funds, grants, or other support were received during the preparation of this manuscript.

Competing interests. The authors declare no competing interests.

Author contributions. Abdullah Yahya Abdullah Omaisani and Ibrahim Sheikh Mohamed contributed equally to the conception, methodology, implementation, experimentation, analysis, and writing of the manuscript. Both authors reviewed and approved the final manuscript.

Data availability. The demonstration data supporting the findings of this study are available from the corresponding author upon reasonable request.

Code availability. The code used for fine-tuning and deployment is available from the corresponding author upon reasonable request.

Ethics approval and consent to participate. Not applicable.

Consent for publication. Not applicable.

References

- [1] Shukor, M., Aubakirova, D., Capuano, F., Kooijmans, P., Palma, S., Zouitine, A., Aractingi, M., Pascal, C., Russi, M., Marafioti, A., Alibert, S., Cord, M., Wolf, T., Cadene, R.: SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics (2025). <https://doi.org/10.48550/arXiv.2506.01844> . <https://arxiv.org/abs/2506.01844>
- [2] Kim, M.J., Pertsch, K., Karamcheti, S., Xiao, T., Balakrishna, A., Nair, S., Rafailov, R., Foster, E., Lam, G., Sanketi, P., Vuong, Q., Kollar, T., Burchfiel, B., Tedrake, R., Sadigh, D., Levine, S., Liang, P., Finn, C.: Openvla: An open-source vision-language-action model. arXiv preprint arXiv:2406.09246 (2024) <https://doi.org/10.48550/arXiv.2406.09246>
- [3] Collaboration, O.X.-E., *et al.*: Open x-embodiment: Robotic learning datasets and rt-x models. arXiv preprint arXiv:2310.08864 (2023) <https://doi.org/10.48550/arXiv.2310.08864>
- [4] Brohan, A., Brown, N., Carbajal, J., *et al.*: Rt-2: Vision-language-action models transfer web knowledge to robotic control. arXiv preprint arXiv:2307.15818 (2023) <https://doi.org/10.48550/arXiv.2307.15818>
- [5] Octo Model Team, Ghosh, D., Walke, H., Pertsch, K., Black, K., Mees, O., Dasari, S., Hejna, J., Xu, C., Luo, J., Kreiman, T., Tan, Y.L., Chen, L.Y., Sanketi, P., Vuong, Q., Xiao, T., Sadigh, D., Finn, C., Levine, S.: Octo: An open-source generalist robot policy. In: Proceedings of Robotics: Science and Systems, Delft, Netherlands (2024). <https://octo-models.github.io/>
- [6] Hu, E.J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W.: Lora: Low-rank adaptation of large language models. In: International Conference on Learning Representations (2022). <https://openreview.net/forum?id=nZeVKeeFYf9>

- [7] Dettmers, T., Pagnoni, A., Holtzman, A., Zettlemoyer, L.: Qlora: Efficient fine-tuning of quantized llms. arXiv preprint arXiv:2305.14314 (2023) <https://doi.org/10.48550/arXiv.2305.14314>
- [8] Dettmers, T., Lewis, M., Belkada, Y., Zettlemoyer, L.: Llm.int8(): 8-bit matrix multiplication for transformers at scale. *Advances in Neural Information Processing Systems* **35** (2022)
- [9] Cadene, R., Aliberts, S., et al.: LeRobot: State-of-the-Art AI for Real-World Robotics. Hugging Face (2024). <https://github.com/huggingface/lerobot>
- [10] Zhao, T.Z., Kumar, V., Levine, S., Finn, C.: Learning fine-grained bimanual manipulation with low-cost hardware. In: *Robotics: Science and Systems (RSS)* (2023)
- [11] Zhai, X., Mustafa, B., Kolesnikov, A., Beyer, L.: Sigmoid loss for language image pre-training. *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 11975–11986 (2023) <https://doi.org/10.48550/arXiv.2303.15343>
- [12] Research, M.: Phi-2: The Surprising Power of Small Language Models. Microsoft Research Blog (2023). <https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models>