# *RoboECC*: Multi-Factor-Aware Edge-Cloud Collaborative Deployment for VLA Models

Zihao Zheng[1], Hangyu Cao[2], Jiayu Chen[1], Sicheng Tian[3] Chenyue Li[2], Maoliang Li[1], Xinhao Sun[4], Guojie Luo[1], †Xiang Chen[1]
[1]School of Computer Science, Peking University, Beijing, China
[2]School of Computer Science, South China University of Technology, Guangzhou, China
[3]School of Artificial Intelligence, Beijing Normal University, Beijing, China
[4]School of EECS, Peking University, Beijing, China

***Abstract*— Vision-Language-Action (VLA) models are mainstream in embodied intelligence but face high inference costs. Edge-Cloud Collaborative (ECC) deployment offers an effective fix by easing edge-device computing pressure to meet real-time needs. However, existing ECC frameworks are suboptimal for VLA models due to two challenges: (1) Diverse model structures hinder optimal ECC segmentation point identification; (2) Even if the optimal split point is determined, changes in network bandwidth can cause performance drift. To address these issues, we propose a novel ECC deployment framework for various VLA models, termed *RoboECC*. Specifically, we propose a model-hardware co-aware segmentation strategy to help find the optimal segmentation point for various VLA models. Moreover, we propose a network-aware deployment adjustment approach to adapt to the network fluctuations for maintaining optimal performance. Experiments demonstrate that *RoboECC* achieves a speedup of up to 3.28x with only 2.55%∼2.62% overhead.**

## I. Introduction

Vision-Language-Action (VLA) models have emerged as the mainstream paradigm for Embodied Artificial Intelligence. Via diverse encoders, they fuse multi-modal information, generate concrete actions, and exhibit high task success rates as well as robust generalization in robotic control scenarios [1], [2].

VLA models incur high inference costs, which severely impede their real-world deployment. Unoptimized VLAs only reach an action control frequency of 1∼3 Hz on edge devices (Fig. 1 (a)), far below the ∼30Hz real-time requirement [3], [4]. Even with acceleration methods, this frequency only increases to 10∼15Hz [5] (on consumer-grade GPUs) and drops further on resource-limited edge devices.

Edge-Cloud Collaboration (ECC) is a highly effective deployment strategy for large-scale models [6]. Typically, it deploys a model's backbone on the cloud (with abundant computing resources) and a lightweight segment on edge devices; via high-bandwidth networks, this setup enables collaborative inference [7], [8]. This configuration not only eases edge devices' computing burden but also boosts overall inference speed and service quality.

Effective integration of ECC with VLA models can enable real-time action generation, yet this process is non-trivial. We summarize two key challenges (Fig. 1 (b)). **Challenge 1: Due to the component heterogeneity inherent in VLA models, existing ECC frameworks are suboptimal in model segmentation.** With technological progress, VLA models have developed diverse architectures. Early models relied solely on detokenizers for token-to-action conversion, while later iterations evolved into specialized Action Models (AM) for action generation—including MLPs, LSTMs, diffusion models, and Diffusion Transformers (DiTs). Owing to the distinct computing/bandwidth needs of VLA components and such architectural complexity, existing static ECC frameworks struggle to find the VLA models' optimal segmentation point.

**Challenge 2: In end-cloud scenarios, internet bandwidth is subject to constant fluctuations. Even when an optimal segmentation point is set, these bandwidth variations can still lead to suboptimal latency.** The data transfer volume at the model's optimal split point reflects only ideal bandwidth conditions. Fluctuations in internet bandwidth can increase or decrease network transmission latency, which in turn alters overall latency. This shifts the optimal split point, requiring fine-grained adjustment. Existing ECC frameworks can not achieve these, leaving it a challenge.

In this paper, we first develop some case studies to prove these two challenges and conclude design insights. Based on these, we propose an end-to-end multi-factor-aware ECC framework for optimal VLA deployment, which is named *RoboECC* (Fig. 1 (c)). Specifically, *RoboECC* consists of two parts. First, we introduce a model-hardware co-aware segmentation strategy to help find the optimal segmentation point for various VLA models. We establish an effective hardware model and VLA structure model, and utilize an automatic algorithm to find the segmentation point. Second, we introduce a network-aware adjustment method to adapt to the network bandwidth fluctuations and maintain optimal performance. We build up a parameter-sharing pool for fine-tuning the edge-cloud deployment and finish the corresponding adjustment methods. Overall, our contributions are threefold:

- We develop case studies to conclude the key problems of achieving ECC deployment for various VLA models and bring key insights for the ECC framework design.
- We propose *RoboECC*, a multi-factor-aware ECC deployment framework for VLA models, which contains a model-hardware co-aware segmentation strategy and a network-aware deployment adjustment method.
- We evaluate *RoboECC* through various VLA models and

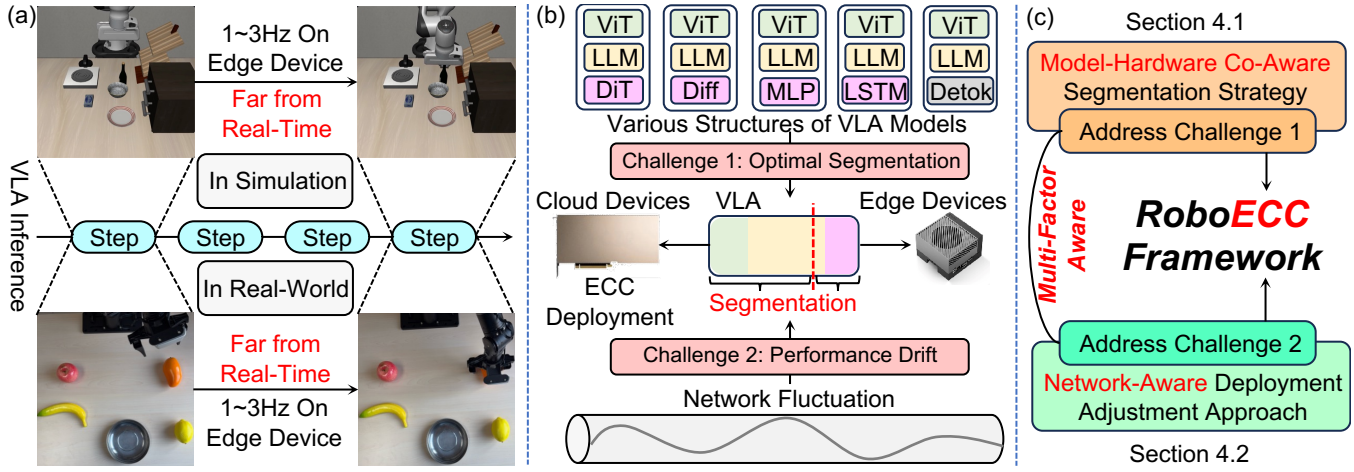


Fig. 1. (a) VLA Inference on Edge Devices (b) Challenges of VLA ECC Deployment (c) Overview of the Proposed *RoboECC* Framework

hardware settings, showing its advantages and evaluating its performance in both simulation and real-world scenarios.

Experiments show that compared with pure edge-side deployment, *RoboECC* achieves a 3.16×∼3.28× speedup on the Orin+A100 platform and a 2.10×∼2.23× speedup on the Thor+A100 platform, with only 2.55%∼2.62% overhead.

## II. Background

### A. VLA Models

*1) **Various Model Structure**:* Traditional VLA models [1], [2] usually consist of three core components: a vision encoder (e.g., Vision Transformer, ViT), a large language model (LLM), and an action detokenizer. With technological progress, modern VLA frameworks have increasingly adopted dedicated action models [9] replace simple detokenizers, enabling more precise action generation. These models (covering diffusion models, Diffusion Transformer (DiT), LSTM, and MLP) also add complexity to VLA structures.

*2) **Inference Cost and Real-Time Requirement**:* The massive parameters and computations of VLA models result in prohibitive inference overhead, failing to meet real-time control demands [10]. Robotic real-time control requires an action generation frequency exceeding 30 Hz, yet unoptimized VLA models only yield 3∼5 Hz, which is drastically below the real-time threshold [3], [5]. Existing edge devices are barely capable of real-time VLA inference, making ECC deployment the primary solution for practical VLA applications.

### B. Edge-Cloud Collaborative Deployment

*1) **ECC-based Co-Inference**:* Existing ECC deployment frameworks can be broadly categorized into two categories: (1) large-small model co-inference [11], [12] and (2) computation offloading [7], [8]. For large-small model co-inference, common techniques are early-exit mechanism and knowledge distillation. However, for VLAs, which are characterized by immense size and complexity, it is hard to construct a model supporting early-exit or distilling knowledge.

*2) **Computing Offloading for Deployment**:* For computation offloading, the two primary methods are parallelism design [8] and model partitioning [7]. Compared with large-small co-inference methods, computing offloading technologies are more suitable for VLA models. Therefore, the ECC framework discussed in this paper refers to the computing offloading category. However, existing ECC frameworks are suboptimal when integrating with VLA models due to the aforementioned reasons in Section I.

## III. Observation and Analysis

In this section, we will analyze in detail why existing ECC frameworks do not fit into the VLA model and provide insights on how to design novel ECC frameworks.

### A. Optimal Segmentation Point under Various Structures

*Motivation ①: The variable structure of VLAs hinders existing ECC frameworks from determining optimal model segmentation points, undermining ECC deployment performance.*

An important part of ECC co-deployment is determining how the original model should be segmented to determine how much load edge-side and cloud-side should bear. We select two VLA models (OpenVLA [2] and CogACT [9]) with different structures and profile them. OpenVLA's model structure contains ViT encoders and LLM, without a generative action model. CogACT's model structure contains ViT encoders, LLM, and a Diffusion-based action model. We start at the last layer of the model, gradually advance the segmentation points, and record the cloud-side/edge-side latency and network communication latency, shown in Fig. 2.

As shown in the Fig. 2, OpenVLA does not have an Action Model, so the end structure is an isomorphic LLM Block, and the change in latency is relatively linear. In this case, given a cloud load budget, the optimal segmentation point can be determined as follows: the block closest to the cloud load budget is near the optimal segmentation point. However, this approach does not work for other VLA architectures such as CogACT. As shown in the Fig. 2, although the 16$^{th}$ block is also under the cloud-side load budget, its latency is significantly higher than the 13$^{th}$ block. In this case, the

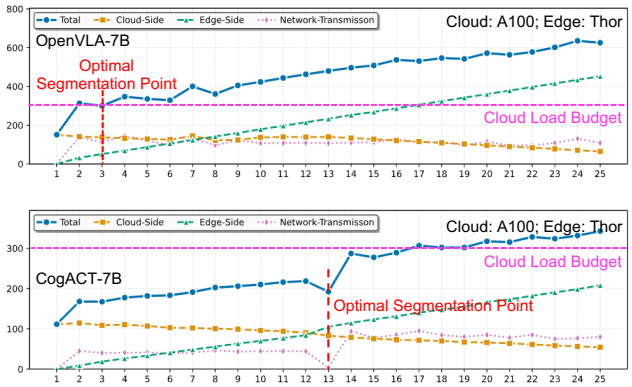


Fig. 2. Latency of Model Segmentation under Various Structures

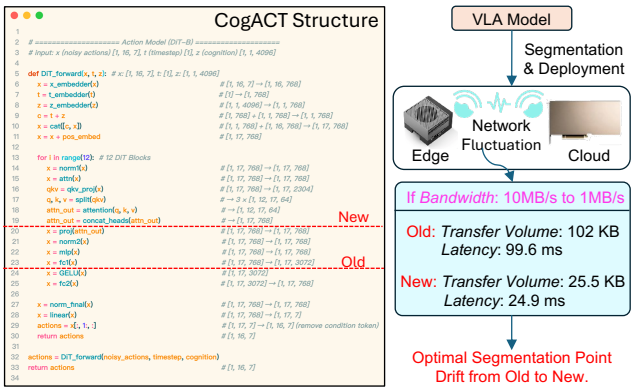


Fig. 3. Performance Drift under Network Bandwidth Fluctuation

optimal split point should be the 13$^{\text{th}}$ block. Therefore, the optimal segmentation point search strategy of VLA needs to be able to adapt to various model structures.

Fortunately, latency increases linearly within the same structure but changes abruptly at structural transitions. This suggests two key considerations for latency prediction: models can accurately forecast latency when targeting hardware within a single structure, and they must also incorporate the model structure itself while accounting for structural changes.

*Insight ①: While VLA's diverse structure hinders segmentation point identification, its model structure features certain repeatability, enabling the estimation of a given layer's hardware execution overhead using hardware performance data.*

### B. *Performance Drift under Network Bandwidth Fluctuation*

*Motivation ②: Network fluctuations can alter transmission delays between the edge and cloud sides, leading to the drift of the optimal segmentation point.*

In ECC deployment, cloud-side and end-side devices connect via a network, while edge-side and cloud-side devices transmit intermediate activations through the network. Network bandwidth constitutes a non-negligible component of latency, and in certain cases, the proportion of network latency can exceed 40%. However, network bandwidth fluctuates in practical scenarios, leading to a drift in the optimal segmentation point. An example is provided in Fig. 3. Assume that "Old" is the optimal segmentation point without considering network fluctuations. For this segmentation point, the amount of data to be transmitted is $[1, 17, 3072]$ (102 KB). When the network condition is good (e.g., 10 MB/s), the latency of this segment is only 9.9 ms. However, if network fluctuations occur (e.g., a drop from 10 MB/s to 1 MB/s), the latency of this segment increases to 99.6 ms, becoming a bottleneck in edge-cloud deployment. Therefore, the optimal segmentation point shifts to "New" because the transmission volume at this point decreases to $[1, 17, 768]$ (25.5 KB), with network latency reduced to 24.9 ms and almost no changes in computing latency at both the edge-side and cloud-side.

Existing ECC frameworks designed for LLMs assume that the network transmission delay is static when determining the optimal segmentation point of the model. This assumption is reasonable for traditional LLMs, as they typically generate relatively long output sequences, which effectively averages out fluctuations in network bandwidth without causing adverse impacts. Additionally, traditional LLMs do not require multi-step inference and perform only one inference per user request.

Yet, the scenario changes for embodied VLA models. First, the output of the VLA model is actions, each of which is a very short sequence, thus having no average effect on network fluctuations. Second, the VLA model relies on multi-step inference to complete the entire task; significant delays in critical steps caused by network fluctuations may lead to severe consequences. Taking the VLA applied in autonomous driving as an example [13], delays in the generation of critical actions, such as braking, caused by network fluctuations, may lead to traffic accidents. Thus, the VLA model is highly sensitive to performance drifts caused by network fluctuations, requiring targeted design when deployed in ECC.

*Insight ②: The ECC framework for the VLA model requires awareness of network fluctuations and dynamic adjustment of the optimal segmentation point.*

## IV. *RoboECC* Framework

This section will detail the proposed deployment framework *RoboECC*, which contains two components: (1) model-hardware co-aware segmentation strategy and (2) network-aware deployment adjustment approach.

### A. *Model-Hardware Co-Aware Segmentation Strategy*

*1) **Structure Modeling of Various VLA Models**:* We first built a structure modeling for VLA models. We divide the VLA models into three sets, namely encoder $\mathbb{S}_{\text{enc}}$, backbone $\mathbb{S}_{\text{bac}}$, and decoder $\mathbb{S}_{\text{dec}}$. For a given VLA model, its structure can be expressed as $[\mathbb{S}_{\text{enc}}, \mathbb{S}_{\text{bac}}, \mathbb{S}_{\text{dec}}]$. Then, we specify that: $\mathbb{S}_{\text{enc}} \in \{\mathcal{M}_{\text{ViT}}\}$; $\mathbb{S}_{\text{bac}} \in \{\mathcal{M}_{\text{LLM}}\}$ and $\mathbb{S}_{\text{dec}} \in \{\mathcal{M}_{\text{De-tokenizer}}, \mathcal{M}_{\text{MLP}}, \mathcal{M}_{\text{LSTM}}, \mathcal{M}_{\text{Diff}}, \mathcal{M}_{\text{DiT}}\}$. These cover all current VLA model structures.

Further, we test the fine-grained layer category $L_i$ in each type of structure, the hidden size of each layer $H_i^L$, and the number of parameters $W_i$ to establish a mapping. We actually measure the total amount of computation and data movements required for each of these structures, represented as $\mathcal{C}_{compute}^{type}$ (FLOPs) and $\mathcal{C}_{datamove}^{type}$ (KB). When the model structure and

details are clear, searching based on this mapping can obtain the cost details for each part, shown in Eq. 1.

$$\text{Search in } \mathcal{M}_{type}(L_i, H_i^L, W_i) \rightarrow (\mathcal{C}_{compute}^{type}, \mathcal{C}_{datamove}^{type}). \tag{1}$$

*2) **Hardware Modeling and Latency Computation**:* However, it is not enough to obtain latency only by the structure modeling, because the latency is also related to hardware performance. Therefore, we propose a computational modeling for both edge-side/cloud-side GPU hardware performance. Considering GPUs' pipeline design, the theoretical latency can be calculated by Eq. 2.

$$\begin{aligned} T_{GPU} &= \sum_{i=0}^{L_i} max(T_{compute}^{Li}, T_{memory}^{Li}) \\ &= \sum_{i=0}^{L_i} max(\frac{\mathcal{C}_{compute}^{L_i}}{P_i \times Parallel_i}, \frac{\mathcal{C}_{datamove}^{L_i}}{Bandwidth_i}), \end{aligned} \tag{2}$$

where $L_i$ is the layer type, $P_i$ is the computing power of the specified GPU, and $Bandwidth_i$ indicates the hardware bandwidth between the GPU and the global memory. To date, structural and hardware modeling are finished, with both utilized for calculating latency.

*3) **Optimal Model Segmentation**:* Based on the aforementioned modeling, we can automatically determine the optimal segmentation point of the model. We perform a depth-first search in a given search space. Note that the search space is determined by the load budget given by the cloud device. The overall search process is shown in Alg. 1. Specifically, we start from the last layer and identify the optimal segmentation point within the allowable cloud-side load range to minimize overall computational delay. We do not integrate a more advanced search strategy into the algorithm because all computational data is derived from our modeling rather than actual measurements or fitting data. This results in extremely low computational load for the search algorithm, with negligible overhead that requires no additional optimization.

### B. Network-Aware Deployment Adjustment Approach

*1) **Network Fluctuation Predictor**:* To dynamically perceive changes in network bandwidth, we design network

**Algorithm 1** Automatic Searching Algorithm to Find the Optimal Model Segmentation Point

**Require:** Edge-Side GPU Latency $T_{GPU}^{Edge}(L_i)$, Cloud-Side $T_{GPU}^{Cloud}(L_i)$, Cloud Offload Budget $B^{Cloud}$, Model Segmentation Point $S$, Number of Layers $n$.
**Ensure:** Cloud Offload $load^{Cloud}(\mathcal{C}_{compute}^{L_i}; \mathcal{C}_{memory}^{L_i})$.

```
for Each S do T_total^S = Σ_0^S T_GPU^Edge(L_i) + Σ_S^n T_GPU^Cloud(L_i)
    if Σ_S^n load^cloud(C_compute^{L_i}; C_memory^{L_i}) < B^cloud then
        S = S + 1
    else break
    end if
    if T_total^S < T_total^optim then
        Optim Point: P_optim = S and T_total^optim = T_total^S
    else Maintain P_optim
    end if
end for
```

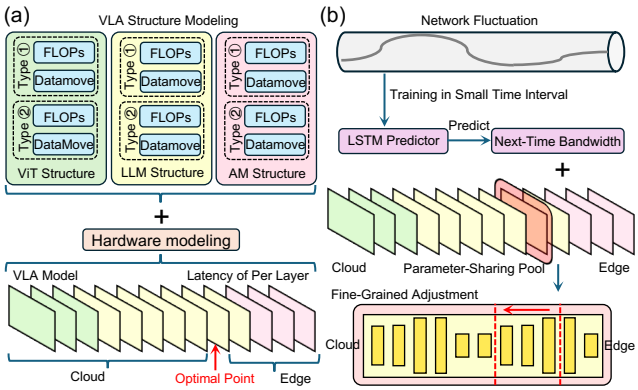


Fig. 4. Components in *RoboECC*: (a) Model-Hardware Co-Aware Segmentation Strategy (2) Network-Aware Deployment Adjustment Approach

fluctuation predictors simultaneously on the cloud and edge sides. Long-term historical data of network bandwidth between the edge and cloud sides is collected to train a lightweight LSTM network. It is important to note that, as our requirement is to predict network bandwidth in real time, the LSTM model needs to be trained with a very fine granularity. It is necessary to ensure that the time interval of input data $t_{input}$ in the LSTM is shorter than the minimum delay of each component after VLA segmentation, as shown in Eq. (3).

$$\{t_{\text{input}} \mid LSTM_{train}\} < min(t_{\text{cloud}}, t_{\text{edge}}) \tag{3}$$

*2) **Parameter-Sharing Pool**:* To enable network-aware tuning, dynamic adjustments to parameters and model structure are essential. However, transmitting these parameters over the network would introduce substantial latency. To address this challenge, we propose an innovative parameter-sharing pool. Specifically, we store all layers of the block containing the optimal segmentation point on both the cloud and edge sides. Since only a single block is involved, the weight overhead for storing this component remains relatively low. Taking the LLAMA Block as an example, the parameters of a layer occupy merely approximately 386 MB. For the cloud, the primary load lies in computational/runtime overhead. Therefore, 386 MB of memory is negligible in this context. Meanwhile, modern edge-side devices are typically equipped with large memory capacities (e.g., Orin with 32 GB / 64 GB and Thor with 64 GB / 128 GB), which are rarely fully utilized by VLA models (for instance, OpenVLA consumes only around 16 GB of memory). Thus, placing these sharing parameters on the edge and cloud sides does not incur significant overhead.

*3) **Fine-Grained Segmentation Adjustment**:* We calculate the variation of network bandwidth $\Delta NB = NB_{t+1}^{\text{pre}} - NB_t^{\text{real}}$ based on the next-moment network bandwidth predicted by LSTM. We set two thresholds $\mathcal{T}_{high}$ and $\mathcal{T}_{low}$ to determine whether dynamic adjustments are required. When $\Delta NB$ is higher than $\mathcal{T}_{high}$, the network bandwidth increases; within the parameter-sharing pool, we search for the layer with the maximum amount of data to be transmitted and adjust the segmentation point to this layer to maximize the utilization of network bandwidth. When $\Delta NB$ is lower than $\mathcal{T}_{low}$, the network bandwidth decreases; in this case, we search for the

layer with the minimum amount of data to be transmitted within the parameter-sharing pool and adjust the segmentation point to this layer to minimize the data transmission volume. These fine-grained adjustments do not require considering changes in computational load on the edge or cloud side, as their impacts on both sides are negligible.

## V. Experiments

### A. Setup

*1) **Hardware Platform**:* We select the Nvidia A100 GPU as the cloud-side device and the Nvidia AGX Jetson Orin/Thor as edge-side devices as platforms, the details of which are shown in Tab. I. The power consumption modes of Orin and Thor are uniformly set to MAX_N mode, as VLA models require high real-time performance, and the maximum power consumption mode can reduce latency.

*2) **Models**:* We select OpenVLA [2] as the evaluation model. It generates action sequences token-by-token directly through its language model backbone, without employing independent action modules. OpenVLA provides a controlled baseline to isolate and evaluate temporal-dynamic sensitivity during sequential inference.

*3) **Baselines**:* Since existing ECC frameworks are not suitable for VLA model deployment, we use device-side-only deployment and cloud-side-only deployment as the baselines. Moreover, we establish a fixed segmentation scheme (with the load approximately equally distributed between the edge and cloud sides) and compare it with *RoboECC*. Comparison with these baselines well demonstrates the advantages of *RoboECC*.

*4) **Evaluation Metrics**:* We adopt edge-side latency, cloud-side latency, total latency, cloud-side load, and edge-side load as the core evaluation metrics. Notably, *RoboECC* does not affect the task success rate, which is therefore not reported.

### B. Main Results

*1) **Results in Simulation Benchmark**:* Tests are conducted in the LIBERO [14] and SimplerEnv [15] simulation benchmark. We conduct 10 tests on each simulation task and report the average inference latency of each step. The results are shown in Tab. II (OpenVLA) and Tab. III (CogACT). Note that total latency includes network latency. Compared with pure edge-side deployment, *RoboECC* achieves a 3.16×∼3.28× speedup on the Orin+A100 platform and a 2.10×∼2.23× speedup on the Thor+A100 platform. Compared with pure cloud-side deployment, *RoboECC* reduces cloud-side load effectively without introducing excessive inference latency. Moreover, compared with fixed segmentation schemes, *RoboECC* exhibits significantly better edge-cloud trade-offs. These results show the superiority of *RoboECC*.

TABLE I
Details of Cloud/Edge-Side Hardware Platform.

| | Cloud-Side | Edge-Side | |
|---|---|---|---|
| **Platform** | A100 GPU | Jetson Orin | Jetson Thor |
| **Frequency** | 1410 MHz | 1300 MHz | 1386 MHz |
| **Computing Power (4-bit)** | 2496 TFLOPs | 275 TFLOPs | 517.5 TFLOPs |
| **Tensor Core Num.** | 432 | 64 | 96 |
| **Memory Bandwidth** | 2039 GB/s | 204.8 GB/s | 273 GB/s |

TABLE II
*RoboECC*'s Performance under OpenVLA LIBERO

| OpenVLA + LIBERO in *RoboECC* | | | | | | | |
|---|---|---|---|---|---|---|---|
| **HW** | **Methods** | **Cloud-Side** | | **Edge-Side** | | **Total** | |
| | | **Lat.** | **Load** | **Lat.** | **Load** | **Lat.** | **Load** |
| Orin | Edge-Only | – | – | 1119.4ms | 14.1GB | 1119.4ms | 14.1GB |
| | Cloud-Only | 151.2ms | 14.1GB | – | – | 151.2ms | 14.1GB |
| | Fixed Seg | 87.9ms | 6.3GB | 717.8ms | 7.9GB | 923.3ms | 14.2GB |
| | *RoboECC* | 136.7ms | 12.1GB | 94.5ms | 2.0GB | 354.4ms | 14.1GB |
| Thor | Edge-Only | – | – | 628.9ms | 14.1GB | 628.9ms | 14.1GB |
| | Cloud-Only | 151.2ms | 14.1GB | – | – | 151.2ms | 14.1GB |
| | Fixed Seg | 89.5ms | 6.3GB | 378.4ms | 7.9GB | 587.2ms | 14.2GB |
| | *RoboECC* | 137.1ms | 12.1GB | 51.3ms | 2.0GB | 300.1ms | 14.1GB |

*2) **Results in Real-World Environment**:* We conduct tests on real robotic systems using OpenVLA. We use the AgileX PIPER robotic arm and Orin+A100/Thor+A100 as the platform. We collect 1,000 data samples for model fine-tuning, following which the CogACT and OpenVLA are deployed using the *RoboECC* framework. A practical example is presented in Fig. 5. When this task is executed, the inference latency of edge-only deployment is approximately 1274.4 ms (Orin) and 667.2 ms (Thor). When integrating *RoboECC*, it only needs 361.8 ms (Orin+A100) and 321.6 ms (Thor+A100), making the movement of the robotic arm smoother and reducing mechanical jitter. The proposed *RoboECC* adapts to real-world scenarios, demonstrating robustness.

*3) **Ablation Studies**:* Ablation experiments are conducted on the Orin+A100 platform, with the results presented in Tab. IV. The integration of co-aware segmentation leads to a significant reduction in overall latency, attributed to the performance gains provided by the cloud side. Incorporating network-aware adjustment results in negligible changes in cloud-side and edge-side latency but a notable decrease in network latency, thereby further lowering the total latency.

### C. Discussion

*1) **Overhead**:* In *RoboECC*, a lightweight LSTM is employed as the network bandwidth predictor, which measures merely 20.1 MB in size. This overhead is negligible compared

TABLE III
*RoboECC*'s Performance under CogACT+SimplerEnv

| CogACT + SimplerEnv in *RoboECC* | | | | | | | |
|---|---|---|---|---|---|---|---|
| **HW** | **Methods** | **Cloud-Side** | | **Edge-Side** | | **Total** | |
| | | **Lat.** | **Load** | **Lat.** | **Load** | **Lat.** | **Load** |
| Orin | Edge-Only | – | – | 775.3ms | 14.5GB | 775.3ms | 14.5GB |
| | Cloud-Only | 111.4ms | 14.5GB | – | – | 111.4ms | 14.5GB |
| | Fixed Seg | 46.9ms | 7.9GB | 437.2ms | 6.6GB | 572.5ms | 14.5GB |
| | *RoboECC* | 81.9ms | 12.0GB | 143.2ms | 2.5GB | 236.1ms | 14.5GB |
| Thor | Edge-Only | – | – | 429.6ms | 14.5GB | 429.6ms | 14.5GB |
| | Cloud-Only | 111.4ms | 14.5GB | – | – | 111.4ms | 14.5GB |
| | Fixed Seg | 47.2ms | 7.9GB | 240.4ms | 6.6GB | 375.4ms | 14.5GB |
| | *RoboECC* | 82.7ms | 12.0GB | 105.7ms | 2.5GB | 192.7ms | 14.5GB |

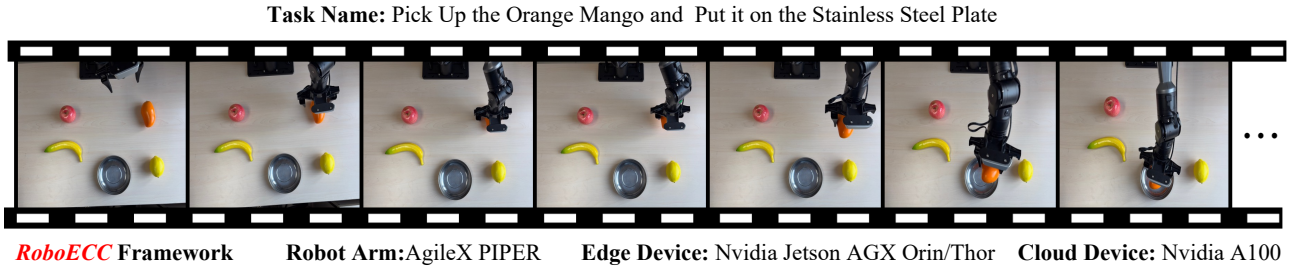


Fig. 5. An Example of *RoboECC* Deployment in Real-World Scenarios

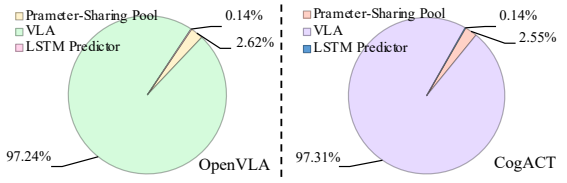


Fig. 6. Overhead of the proposed *RoboECC* Framework

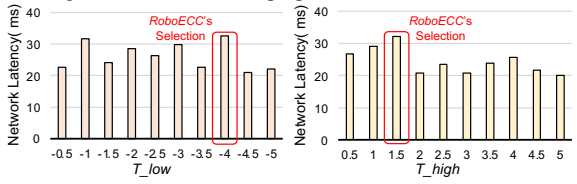


Fig. 7. Hyper-parameters $\mathcal{T}_{low}$ and $\mathcal{T}_{high}$ in *RoboECC*

to that of the VLA model. As illustrated in Fig. 6, the parameter-sharing pool accounts for only 2.55%∼2.62% of the total overhead, while the LSTM-related overhead is negligible. We test the time overhead of split point adjustment under different network bandwidths, and the results show that its average value is only 10.7 ms, while the average latency reduction achieved is 32.6 ms, indicating that the overhead of this part is far less than the gain.

*2) **Hyper-Parameters**:* The hyper-parameters $\mathcal{T}_{low}$ and $\mathcal{T}_{high}$ are determined using the following method: We take the maximum $\Delta NB$ in historical network bandwidth changes as $\mathcal{T}_{high}$, then conduct experiments to determine $\mathcal{T}_{low}$. After $\mathcal{T}_{low}$ is determined, we further perform experiments to update $\mathcal{T}_{high}$. The results of this process are illustrated in Fig. 7.

*3) **Scope**:* In *RoboECC*, we only provide a hardware model for GPUs, and have not yet considered accelerators with other architectures, which are regarded as future work.

## VI. Conclusion

We propose a multi-factor-aware ECC deployment framework for VLA models, named *RoboECC*. In *RoboECC*, we propose a model-hardware co-aware segmentation strategy and a network-aware deployment adjustment approach for optimal performance. Experiments show that *RoboECC* achieves a speedup of up to 3.28x with only 2.55%∼2.62% overhead.

TABLE IV
Ablation Studies of *RoboECC*.

| Methods | Cloud-Side | Edge-Side | Total |
|---|---|---|---|
| **Edge-Only** | – | 1119.4ms | 1119.4ms |
| **+ Co-Aware Segmentation** | 135.9ms | 99.8ms | 392.7ms |
| **+ Network-Aware Adjustment** | 136.7ms | 94.5ms | 354.4ms |

## References


[1] B. Zitkovich, T. Yu, S. Xu, P. Xu, T. Xiao, F. Xia, J. Wu, P. Wohlhart, S. Welker, A. Wahid *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," in *Conference on Robot Learning*. PMLR, 2023, pp. 2165–2183.

[2] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi *et al.*, "Openvla: An open-source vision-language-action model," *arXiv preprint arXiv:2406.09246*, 2024.

[3] Z. Zheng, Z. Mao, M. Li, J. Chen, X. Sun, Z. Zhang, D. Cao, H. Mei, and X. Chen, "Kerv: Kinematic-rectified speculative decoding for embodied vla models," *arXiv preprint arXiv:2603.01581*, 2026.

[4] Z. Zheng, H. Cao, S. Tian, J. Chen, M. Li, X. Sun, H. Zou, Z. Zhang, X. Liu, D. Cao *et al.*, "Dyq-vla: Temporal-dynamic-aware quantization for embodied vision-language-action models," *arXiv preprint arXiv:2603.07904*, 2026.

[5] Z. Zheng, Z. Mao, S. Tian, M. Li, J. Chen, X. Sun, Z. Zhang, X. Liu, D. Cao, H. Mei *et al.*, "Heisd: Hybrid speculative decoding for embodied vision-language-action models with kinematic awareness," *arXiv preprint arXiv:2603.17573*, 2026.

[6] Z. Zheng, S. Tian, H. Cao, C. Li, J. Chen, M. Li, X. Sun, H. Zou, G. Luo, and X. Chen, "Rapid: Redundancy-aware and compatibility-optimal edge-cloud partitioned inference for diverse vla models," *arXiv preprint arXiv:2603.07949*, 2026.

[7] M. Zhang, X. Shen, J. Cao, Z. Cui, and S. Jiang, "Edgeshard: Efficient llm inference via collaborative edge computing," *IEEE Internet of Things Journal*, 2024.

[8] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "Coedge: Cooperative dnn inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, 2020.

[9] Q. Li, Y. Liang, Z. Wang, L. Luo, X. Chen, M. Liao, F. Wei, Y. Deng, S. Xu, Y. Zhang *et al.*, "Cogact: A foundational vision-language-action model for synergizing cognition and action in robotic manipulation," *arXiv preprint arXiv:2411.19650*, 2024.

[10] R. Zhang, M. Dong, Y. Zhang, L. Heng, X. Chi, G. Dai, L. Du, Y. Du, and S. Zhang, "Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation," *arXiv preprint arXiv:2503.20384*, 2025.

[11] S. Laskaridis, S. I. Venieris, M. Almeida, I. Leontiadis, and N. D. Lane, "Spinn: Synergistic progressive inference of neural networks over device and cloud," in *Proceedings of the 26th annual international conference on mobile computing and networking*, 2020, pp. 1–15.

[12] C. Ding, A. Zhou, Y. Liu, R. N. Chang, C.-H. Hsu, and S. Wang, "A cloud-edge collaboration framework for cognitive service," *IEEE Transactions on Cloud Computing*, vol. 10, no. 3, pp. 1489–1499, 2020.

[13] J. Cao, Q. Zhang, P. Jia, X. Zhao, B. Lan, X. Zhang, X. Wei, S. Chen, Z. Li, Y. Wang *et al.*, "Fastdrivevla: Efficient end-to-end driving via plug-and-play reconstruction-based token pruning," *arXiv preprint arXiv:2507.23318*, 2025.

[14] B. Liu, Y. Zhu, C. Gao, Y. Feng, Q. Liu, Y. Zhu, and P. Stone, "Libero: Benchmarking knowledge transfer for lifelong robot learning," *Advances in Neural Information Processing Systems*, vol. 36, pp. 44 776–44 791, 2023.

[15] X. Li, K. Hsu, J. Gu, K. Pertsch, O. Mees, H. R. Walke, C. Fu, I. Lunawat, I. Sieh, S. Kirmani, S. Levine, J. Wu, C. Finn, H. Su, Q. Vuong, and T. Xiao, "Evaluating real-world robot manipulation policies in simulation," *arXiv preprint arXiv:2405.05941*, 2024.