Jetson-PI: Towards Onboard Real-Time Robot Control via Foresight-Aligned Asynchronous Inference

Zebin Yang[1] **Qi Wang**[1] **Yunhe Wang**[1] **Xiurui Guo**[3] **Bo Yu**[2]
Shaoshan Liu[2] **Jiafeng Xu**[1] **Hao Dong**[1] **Meng Li**[1*]
[1]Peking University [2]AIRS [3]PrimeBot Research Institute
* Corresponding author. Email: meng.li@pku.edu.cn

Abstract: Vision-Language-Action (VLA) models have achieved impressive performance on diverse embodied tasks. However, deploying VLA models on low-power onboard devices, such as the Jetson Orin, remains challenging due to their high computational complexity, which leads to substantial inference latency and low control frequency. Asynchronous inference can partially mask this latency by parallelizing action execution and subsequent inference, but it introduces two critical issues: perception-execution misalignment and long reaction time. In this paper, we propose Jetson-PI, a method for efficient VLA deployment on onboard devices via Foresight-Aligned Asynchronous Correction. To address misalignment, we train a lightweight future correction module that predicts future environment representation conditioned on committed actions, enabling the action expert to directly predict actions from the future time step. To reduce reaction time, we introduce confidence-based scheduling optimization that adaptively balances VLM and action expert invocations, complemented by system-level accelerations including CUDA graph reuse, GPU-resident intermediate buffering, and flow unrolling. Extensive experiments demonstrate that Jetson-PI achieves 8.66× and 5.41× improvements in control frequency compared with naive PyTorch and vla.cpp on NVIDIA Jetson Orin, while outperforming VLASH by 14.8% in average success rate on the LIBERO benchmark. The code of our asynchronous algorithm is available on https://github.com/PKU-SEC-Lab/Jetson-PI, and our efficient `llama.cpp`-based inference engine is available on https://github.com/PKU-SEC-Lab/Jetson-PI-Edge.

Keywords: Robotic manipulation, Edge computing, Asynchronous inference

1 Introduction

Based on Vision-Language Models (VLMs), Vision-Language-Action (VLA) models have demonstrated remarkable performance across a wide range of embodied tasks [1, 2]. By leveraging the rich knowledge embedded in pre-trained VLMs and fine-tuning on high-quality robot demonstration datasets, VLA models can directly predict actions chunks from raw observations and natural language instructions [3, 4]. However, the high parameter count and computational complexity of VLA models pose significant challenges for their deployment on real robotic systems, where low-latency inference is critical yet onboard compute resources are often limited [5, 6, 7, 8].

To achieve low inference latency and high control frequency, existing works often deploy VLA models on high-end GPUs such as NVIDIA RTX 4090 [1, 3, 9, 10, 11, 12], which provide substantial compute and bandwidth resources. However, these devices introduce severe power consumption issues [13, 14]. As shown in Figure 1(a), using an RTX 4090 can reduce battery life by 6.0× compared to onboard devices like Jetson Orin, significantly limiting robot working time. This limits

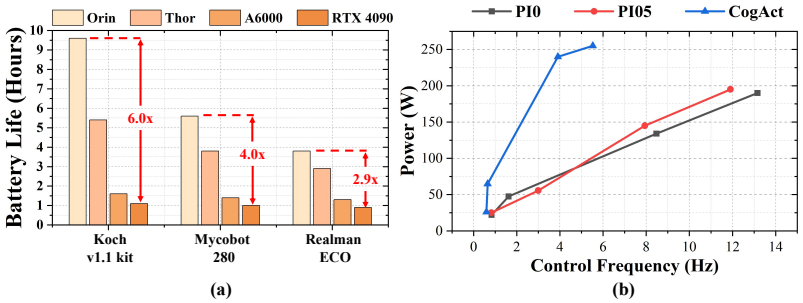

Figure 1: (a) Battery life of three robots equipped with four different computing devices. We use a 500 Wh WILLQ AGV lithium battery pack, and robot's mechanical power consumption is included. (b) Power consumption and control frequency of different VLA inference on four computing devices: Jetson Orin, Jetson Thor, RTX A6000, and RTX 4090.

existing VLA works to laboratory demos and makes it difficult to expand its application scenarios [15, 16, 17]. Even when running such computing devices as online servers, it will also bring extra network latency, and the robot's activity area will be limited [17, 18]. Therefore, deploying VLA models on low-power onboard devices is essential for embodied applications [17, 19, 20, 21, 22].

However, due to the limited resources of onboard devices, VLA models suffer from substantial inference latency [17, 23, 24]. As shown in Figure 1(b), state-of-the-art VLA models such as $\pi_{0.5}$ [3] achieve an inference latency of approximately 1.4 seconds on a Jetson Orin, resulting in a control frequency around 0.7 Hz. This low frequency leads to slow reaction to environmental changes and long pauses between action chunks [25, 26, 27]. Even if asynchronous inference [28, 29, 30] can partially hide inference latency by predicting the next chunk while executing the current chunk, we find that it introduces two critical issues. First, when the VLA finishes predicting an action chunk, the external environment has already changed relative to the image input, leading to a prediction-execution misalignment problem, which becomes even more severe with a long inference time. Second, the robot's reaction speed remains bounded by the VLA inference time, making it difficult to respond in time for complex embodied tasks.

In this paper, we present Jetson-PI, a method for deploying VLA models on low-power onboard devices via foresight-aligned asynchronous inference. To address prediction-execution misalignment under long inference latency, we train a correction module that predicts future VLM representations conditioned on the actions committed for execution. These predicted representations are then provided as future information to the action expert, enabling it to directly predict actions starting from the future time step. This allows the predicted trajectory to better align with the actual environment at execution time. For the slow reaction time issue, we approach the problem from both scheduling and system perspectives. On the scheduling side, as the future correction module can dynamically adjust the foresight horizon, we can invoke the VLM once and subsequently call the action expert multiple times, thus improving control frequency and reducing reaction time. On the systems side, we design an inference framework for VLA on edge platforms based on llama.cpp, and accelerate VLA inference through computation graph reuse and unrolling to cope with the limited bandwidth. We conduct experiments both in simulation and on real robots across multiple edge devices. On Jetson Orin, Jetson-PI achieves 8.66× and 5.41× improvements in control frequency compared with naive PyTorch and vla.cpp [31]. And it outperforms VLASH [10] by 14.8% in average success rate on the LIBERO benchmark [32].

2 Background

Vision-language-action models. VLA models have demonstrated strong generalization across a wide range of embodied tasks. State-of-the-art VLA models, such as π_0 and $\pi_{0.5}$, typically adopt a "VLM + action expert" architecture [1, 3, 33, 34, 35, 36, 37, 38]. During each inference, the current image and language instruction are fed into VLM, which computes and passes the resulting

KV cache to action expert. The action expert takes this context along with randomly initialized action noise and uses flow matching to predict an action chunk through multiple denoising steps. The detailed architecture of π_0 and $\pi_{0.5}$ is shown in Appendix A. These VLA models are typically deployed under synchronous inference, where the robot must wait for the current action chunk to finish execution before the next inference [26]. Consequently, as shown in Figure 2, the long VLA latency of onboard devices results in long robot pauses between action chunks.

Asynchronous VLA Inference. As shown in Figure 2, asynchronous inference eliminates pauses between chunks by parallelizing execution of the current chunk with inference of the next chunk [39, 29, 40]. Among them, SmolVLA implements naive asynchronous inference and directly switches to new action chunks after prediction [28], and RTC mitigates discontinuity between chunks by inpainting the new chunk to produce smoother trajectories [26]. However, they still can not solve the prediction-execution misalignment that the environment has already changed relative to the input image of the chunk prediction, which leads to prediction accuracy drop. VLASH predicts the robot's state at the end of VLA inference ($q_{t+\Delta} = q_t + a_t + ... + a_{t+\Delta-1}$), where Δ is VLA latency in terms of action steps. And uses $q_{t+\Delta}$ to guide the next chunk prediction [10]. But future robot state alone does not reflect environmental changes, and its effectiveness degrades as inference latency increases. In contrast, we train a future correction module directly predicting future environmental representation. This allows the next chunk prediction to account for how the environment will be affected by the committed actions.

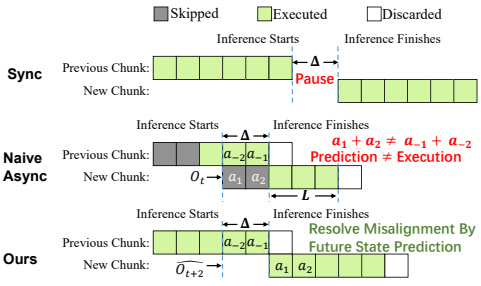

Figure 2: Comparison of different inference modules. Δ is VLA latency in terms of action steps. L is the number of actions executed in each chunk.

World Action Models. Another line of work related to ours is World Action Models (WAMs), which leverage world modeling, i.e., predicting future environment, to guide action prediction [41, 40, 42, 43, 44]. [45, 46] explicitly predict future images, and [41, 47, 48, 49] predict latent representations of future states. However, WAMs typically predict environments at the granularity of entire action chunks or sub-tasks, which is usually used as the desired final state and assists in predicting the current action [40, 9]. Our future correction module can be viewed as a lightweight "world model". The distinction is that our method can dynamically adjust the foresight horizon at action step granularity, which can predict the environment representation after executing the committed actions and assist in predicting the following actions. This enables addressing the perception-execution misalignment across different computing devices with varying VLA inference latencies.

3 When Asynchronous Inference Meets Onboard Computation

We conduct a detailed analysis of asynchronous VLA inference and identify two critical challenges that limit onboard deployment: perception-execution misalignment and long reaction time.

Perception-execution Misalignment. Perception-execution misalignment arises because the environment continues to evolve while the VLA model performs inference for the next action chunk. As shown in Figure 3(a)(b), for naive asynchronous and RTC, with VLA inference latency increases, perception-execution misalignment becomes more severe, leading to task success rate drop. While VLASH maintains high accuracy with low Δ, for longer latency, predicting future robot state alone is insufficient to capture environmental changes. Therefore, a method that can provide future environmental information for asynchronous inference on onboard devices is needed.

Long Reaction Time. Reaction time refers to the interval between an environmental change and the robot's corresponding response. For asynchronous inference, as shown in Figure 4, the reaction time in terms of action steps ranges from Δ to $\Delta + L$ [50, 51], where L is the number of actions executed in each chunk ($L = 3$ in Figure 2(b)). As shown in Figure 3(c)(d), increasing L under varying Δ

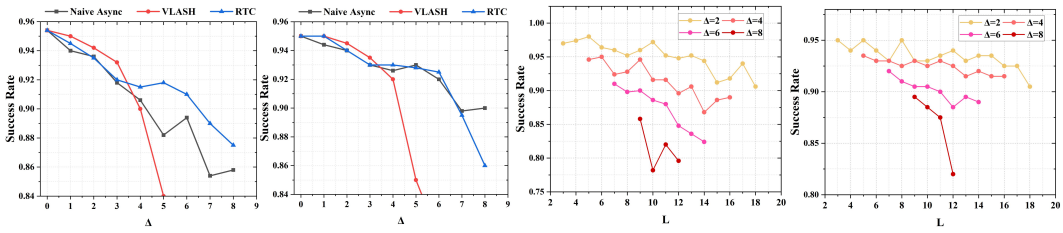

Figure 3: (a)(b) Success rate changes with different Δ on LIBERO-Spatial and LIBERO-Goal using π_0. Here, $L = H - \Delta$, where $H = 20$ is the action chunk size. (c)(d) Success rate changes with different L values. using naive asynchronous inference on LIBERO-Spatial and LIBERO-Goal.

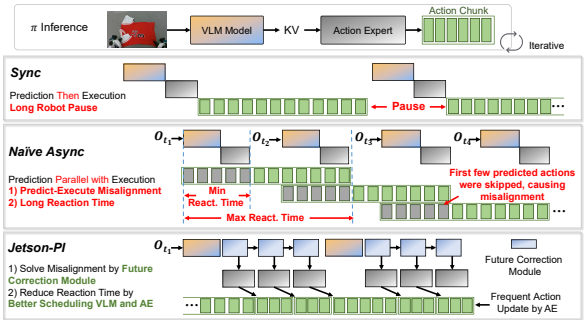

Figure 4: Overview of Jetson-PI.

VLA	Device	ViT	LLM	Action Expert	Total
π_0	Orin-30W	322.4	856.4	1090.0	2268.8
	Orin-50W	152.5	632.3	518.2	1403.0
	Thor	63.7	185.0	199.2	447.9
	A6000	23.4	43.5	46.9	116.1
	4090	13.1	32.2	22.4	69.5
$\pi_{0.5}$	Orin-30W	329.0	858.2	1102.2	2289.4
	Orin-50W	152.3	631.0	536.8	1420.8
	Thor	61.7	187.0	209.2	457.9
	A6000	23.8	51.2	50.2	128.2
	4090	13.2	35.9	25.2	76.4

Table 1: Inference latency breakdown (ms) of different VLA models on various devices. We consider different power modes of Jetson Orin.

leads to a drop in task success rate. Because a larger L results in a longer average reaction time. In addition, later actions in a predicted chunk tend to be lower quality, as the environment becomes increasingly uncertain when these actions are executed [50]. To reduce L without introducing pauses between chunks, a straightforward approach is to perform continuous VLA inference (setting $L = \Delta$). However, the reaction time is still bounded by Δ. Consequently, the objective of reducing reaction time is transformed into lowering inference latency, especially for onboard deployment.

Overview. Figure 4 presents an overview of Jetson-PI. To address prediction-execution misalignment, we propose Foresight-Aligned Asynchronous Correction, which adaptively predicts future environment representation conditioned on the committed actions. This allows the action expert to directly predict actions based on the future environment rather than the outdated observation. For the long reaction time problem, on scheduling side, we propose Confidence-based Scheduling Optimization, which dynamically adjusts the invocation frequency of the VLM and the action expert based on the confidence of the future environment prediction. On systems side, we conduct computation graph reuse, intermediate buffering, and unrolling to accelerate VLA inference.

4 Method

4.1 Foresight-Aligned Asynchronous Correction

Requirements for the future correction module. To address the prediction-execution misalignment problem, we train a future correction module that predicts the environment representation at the time when VLA inference completes, guiding next chunk prediction. However, the future correction module must satisfy the following requirements: (1) Action-conditioned prediction: The module must predict future environment states conditioned on the actions committed to be executed, since these actions directly determine how the environment evolves. (2) Lightweight design: The module must introduce minimal additional latency, as extra delay would exacerbate the misalignment problem. (3) Adaptability to varying Δ: as shown in Table 1, VLA inference latency varies

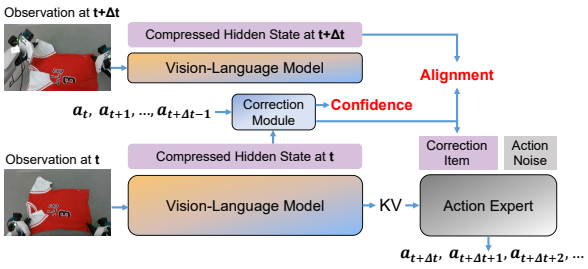

Figure 5: Foresight-Aligned Asynchronous Correction.

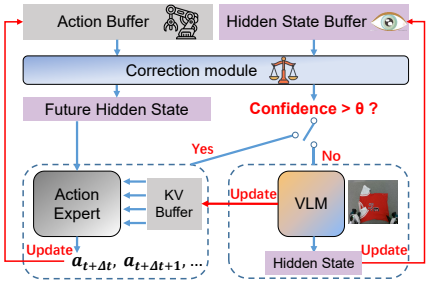

Figure 6: Scheduling Optimization.

across different hardware platforms and even under different power modes on the same device. The module must adapt to a wide range of Δ values to remain effective in diverse deployment scenarios.

The design of Foresight-Aligned Asynchronous Correction is shown in Figure 5. To avoid introducing excessive latency, we do not directly predict future images or correct the KV cache at each layer. Inspired by [48], we take the compressed final-layer output of the VLM at t and the committed actions, then pass them through future correction module that predicts the compressed VLM final-layer output at $t + \Delta$. Then pass this lightweight correction item to action expert, enabling action expert to directly predict actions starting from $t+\Delta$. The parameter size of future correction module is just 40M, which is only 1% of the whole VLA and introduces negligible cost. Architecture details of final-layer state compressor and future correction module are shown in Appendix B.

To ensure that the future correction module effectively benefits action prediction, the training process must address two objectives: (1) enabling action expert to predict actions starting at $t + \Delta$ with correction item, and (2) enabling future correction module to accurately predict the future environment representation. To this end, we design a two-stage Correction-aware Training pipeline. In the first stage, we use the ground-truth VLM final-layer state at time $t + \Delta$ and train the compressor and the action expert using the action loss, allowing them to learn how to extract useful information from the future hidden state. In the second stage, we feed the compressed VLM final-layer output at time t and the action sequence from t to $t + \Delta$ into the future correction module. The module is trained to predict the compressed VLM final-layer output at time $t + \Delta$. Additionally, the module is trained to output a confidence estimate for its future environment prediction, which will later facilitate scheduling optimization. The loss function of the second stage is

$$L_{predict} = \|\hat{h}_{t+\Delta} - h_{t+\Delta}\|_2, \quad L_{total} = L_{predict} + \lambda\|\hat{c} + L_{predict}\|_2, \tag{1}$$

where $\hat{h}_{t+\Delta}$ and $h_{t+\Delta}$ is predicted and ground-truth compressed VLM final-layer output at $t + \Delta$ respectively, $\hat{c}$ is the predicted confidence. During training, we randomly sample Δ to enable the future correction module to adapt to varying inference latencies.

4.2 Confidence-based Scheduling Optimization

Rethinking the role of VLM and action expert. With future correction module, to reduce reaction time, a naive method is to use only the initial image observation, relying solely on the future correction module to predict environmental changes and use the action expert to output all subsequent action chunks without further VLM invocation. However, the prediction of future correction module introduces error. These errors accumulate over time and lead to a drop in the success rate [36]. Here we reconsider the roles of the VLM and the action expert after introducing future correction module. The VLM serves to observe current environment and thereby improve the accuracy of the future correction module's prediction. The action expert is invoked to predict actions in real time. Balancing their invocation frequencies is key to achieving low reaction time while maintaining accuracy.

Based on this, we design a Confidence-based Scheduling Optimization approach, where the future correction module serves as the scheduler. Its predicted confidence determines when to invoke the VLM and the action expert. Specifically, as shown in Figure 6, when the confidence is above a

Table 2: Hardware specifications and roofline analysis of different computing devices.

	RTX 4090	RTX A6000	Orin 32GB	Orin 64GB	Thor
Max Power (W)	450	300	50	50	130
TOPS (FP16 TFLOPS)	309	309	200	275	510
Memory	24GB GDDR6X	48GB GDDR6	48GB LPDDR5	48GB LPDDR5	48GB LPDDR5X
Bandwidth (GB/s)	1024	768	204.8	204.8	273
Balance Computational Intensity (TOPS/Bandwidth $\times 1000$)	618	804	1953	2685	3736
VLM	Compute Bound	Compute Bound	Bandwidth Bound	Bandwidth Bound	Bandwidth Bound
Action Expert	Bandwidth Bound	Bandwidth Bound	Bandwidth Bound	Bandwidth Bound	Bandwidth Bound

threshold θ, we consider the predicted future environment information to be sufficiently accurate. So the VLM will be skipped, and the action expert directly predicts actions, improving real-time responsiveness. When the confidence falls below θ, we invoke the VLM to observe the current environment and update both the KV buffer used by the action expert and the hidden state buffer used by the future correction module for subsequent predictions. In this way, our method adaptively adjusts the invocation frequency of the VLM and the action expert, reducing reaction time without dropping accuracy. We provide an example of confidence evolving over time in experiments and a pseudocode of our algorithm in Appendix C.

Why don't we parallelize VLM and action expert? [52] deploys π_0 on NVIDIA RTX 4090 and parallelizes VLM and action expert inference to reduce reaction time. However, we find this strategy is ineffective for onboard devices. We analyze VLA computation on different devices using the roofline model [41, 53]. The dominant computation in VLA inference is matrix multiplication. For a matrix multiplication operator with input sizes $M \times K$ (activation) and $K \times N$ (weight), the theoretical execution time in FP16 precision is $max(\frac{M*K*N}{compute\ throughput}, \frac{2*(M*K+K*N)}{bandwidth})$, where M represents token length and N represents model embedding dimension. Specifically, the inference latency is compute-bound when the first term dominates and bandwidth-bound when the second term dominates. The value of M when the two terms are equal corresponds to the balanced arithmetic intensity. As shown in Table 2, on a high-end GPU such as the RTX 4090, the VLM is compute-bound while the action expert is bandwidth-bound. Therefore, parallelizing the two fully utilizes both compute and bandwidth resources. In contrast, onboard devices typically have much tighter bandwidth. Even VLM computation can not reach the balanced arithmetic intensity, and both the VLM and the action expert become bandwidth-bound. Running them in parallel leads to severe contention for the shared bandwidth, ultimately causing both to slow down significantly.

4.3 System Design

Unique characteristics of VLA inference. Beyond scheduling optimization, we also accelerate VLA inference through system-level design to further reduce reaction time. To the best of our knowledge, no existing deployment framework is specifically designed for edge-side VLA inference. We build our system upon llama.cpp, a popular inference framework for LLMs edge deployment, and adapt it to the unique characteristics of VLA models, which we identify as follows: (1) **Deterministic token length.** Unlike LLM decoding, where token length grows progressively, given camera configurations and image resolutions, the token length in VLA inference remains largely constant. The only potential variation is the language instruction, which is typically short and exhibits minimal length fluctuation. (2) **Cross-component communication overhead.** A VLA model consists of three components—ViT, LLM, and action expert—that communicate with each other. For example, ViT encodes visual observations and then passes them to LLM, and LLM generates KV caches and passes them to action expert. Traditional inference frameworks write such intermediate results back to CPU memory after GPU computation, introducing unnecessary communication latency. (3) **Repeated action expert invocations.** The action expert adopts a flow matching formulation, which requires multiple denoising steps to produce an action chunk. This repeated invocation of the identical computation graph introduces additional latency.

Table 3: Success rate comparison across different methods on four sub-datasets of LIBERO, using $\pi_{0.5}$. We report the SR of using Foresight-Aligned Asynchronous Correction alone (ours) and with Confidence-based Scheduling Optimization (+Sched). We estimate inference time of action expert as $\Delta_{ae} = \lceil \frac{\Delta}{3} \rceil$., which is a common ratio both on high-end GPUs such as RTX 4090 and on onboard GPUs such as Orin. We only train one model for all Δ values.

	SPATIAL				OBJECT				GOAL				LIBERO-10			
Sync.	97.3				99.6				96.7				93.5			
Δ	VLASH	RTC	Ours	+Sched	VLASH	RTC	Ours	+Sched	VLASH	RTC	Ours	+Sched	VLASH	RTC	Ours	+Sched
1	98.8	97.1	97.7	98.6	99.2	98.5	98.7	98.8	96.7	96.5	97.0	97.5	94.4	92.3	93.4	93.1
2	97.5	95.4	97.1	97.6	99.2	98.5	98.5	99.3	97.0	96.0	96.2	97.3	94.6	92.1	92.7	92.9
3	94.4	94.5	97.1	97.2	98.8	96.4	98.2	98.4	93.3	94.3	96.7	96.4	91.9	89.6	91.3	91.9
4	92.5	92.5	96.7	96.9	96.9	97.7	97.8	98.9	93.3	93.3	95.9	96.9	89.6	85.8	91.8	92.3
5	84.3	91.2	95.9	96.7	94.3	96.9	96.9	98.6	89.9	93.9	96.0	96.1	80.3	83.6	91.5	92.0
6	74.4	91.7	97.0	97.5	88.5	97.4	98.3	98.4	81.3	94.0	96.7	96.9	78.5	84.6	92.2	92.9
7	51.3	91.7	97.0	97.3	81.4	97.1	98.0	98.4	76.1	92.6	97.1	97.2	77.0	85.3	92.6	92.6
8	46.7	90.9	97.2	97.5	65.1	93.8	98.3	98.8	70.8	93.4	96.8	96.8	65.5	83.2	92.3	92.7
9	30.1	88.8	97.0	97.3	51.7	93.1	97.5	97.6	59.7	92.5	96.3	96.5	59.6	81.0	92.0	92.2
Avg	74.4	92.6	97.0	**97.4**	86.1	96.6	98.0	**98.6**	84.2	94.1	96.5	**96.8**	81.3	86.4	92.2	**92.5**

Table 4: Inference latency and control frequency on different devices, evaluated on LIBERO.

Jetson Orin						
Method	ViT (ms)	LLM (ms)	Action Expert (ms)	Total (ms)	Reaction Time (ms)	Control Frequency (Hz)
Naive PI05	152.3	631.0	536.8	1420.8	1420.8	0.70
+Schedule opt.	152.3	631.0	536.8	1420.8	674.9	1.48
+Graph reuse	79.5	212.6	184.0	476.1	227.0	4.41
+Intermediate Buffer & Unroll	79.5	210.3	123.1	412.9	165.1	6.06
Jetson Thor						
Method	ViT (ms)	LLM (ms)	Action Expert (ms)	Total (ms)	Reaction Time (ms)	Control Frequency (Hz)
Naive PI05	61.7	187.0	209.2	457.9	457.9	2.18
+Schedule opt.	61.7	187.0	209.2	457.9	244.7	4.08
+Graph reuse	51.6	159.2	161.3	372.1	191.4	5.22
+Intermediate Buffer & Unroll	51.6	156.2	101.7	309.5	131.8	7.59

Based on these opportunities, we conduct the following optimizations. (1) **Computation graph reuse.** As VLA inference has deterministic token length, we directly reuse the CUDA graph from the first inference to save graph construction time. To ensure that the token length remains constant, we pad the language instruction to a fixed length for every inference. (2) **GPU-resident Intermediate Buffer.** Because the sizes of the intermediate results between different components are deterministic, we can directly reserve fixed-size buffers in GPU memory, avoiding writing them back to CPU memory. (3) **Flow matching unroll.** We unroll the flow matching computation graph, fusing multiple denoising iterations into a unified computation graph. This reduces the number of graph invocation calls. Although unrolling increases the graph construction time, this overhead is incurred only once during the first inference step, and the graph is reused in subsequent inferences. Our method is orthogonal to quantization [54, 55, 56, 57], pruning [58, 59, 60], and other acceleration techniques. Here, we focus on system-level optimizations that do not alter the model computation.

5 Experiments

We conduct comprehensive experiments in both sim and real to validate the following questions: **(1)** How does the performance of Jetson-PI compare to other asynchronous inference methods (RTC and VLASH)? **(2)** How much improvement in reaction time and control frequency does Jetson-PI achieve on onboard devices compared to standard VLA inference? **(3)** Does Jetson-PI generalize across different inference latencies (varying Δ) and different onboard devices (Orin and Thor)?

5.1 Simulation Experiments

Experiment Setup. We evaluate Jetson-PI on LIBERO [32] benchmark based on $\pi_{0.5}$ model. We compare Jetson-PI with three baselines: (1) **Sync.** serves as an optimal baseline and the inferred

Method	Picking	Folding	Placing
RTX 4090 (Baseline)	10/10	7/10	10/10
Jetson Orin (Naive Async)	6/10	0/10	5/10
Jetson-PI (Ours)	10/10	8/10	9/10

Figure 7: Real-world results on 3 subtasks (picking, folding, placing) on different deployments.

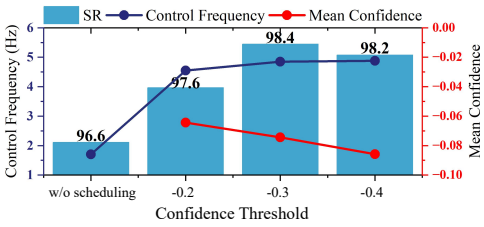

Figure 8: Impact of θ, evaluated on LIBERO-Spatial on Jetson Orin.

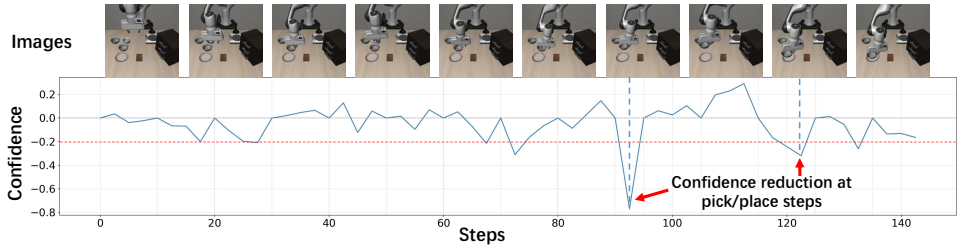

Figure 9: An example of confidence evolution. The confidence threshold is -0.2, and the instruction is "pick up the black bowl between the plate and the ramekin and place it on the plate".

delay is set to 0 [3]. (2) **RTC** freezes the actions committed to execute and inpaints the rest for smoother trajectories [26]. (3) **VLASH** estimates future robot state to guide action prediction [10].

Main Results. The simulation results are shown in Table 3. On average over four sub-datasets and different Δ, our method outperforms VLASH and RTC by 14.8% and 3.9%. Notably, as Δ increases and the perception-execution misalignment problem becomes more severe, both VLASH and RTC exhibit significant performance degradation. In contrast, Jetson-PI maintains consistently high accuracy, as it adaptively provides future environment information for action prediction across varying Δ. For $\Delta = 9$, Jetson-PI shows 45.6% and 7.0% on success rate over VLASH and RTC, on average across four datasets, showing the effectiveness of our method. Compared to synchronous inference, our method eliminates pauses between action chunks with negligible accuracy loss.

5.2 Latency Evaluation

Table 4 presents our latency evaluation. Compared to $\pi_{0.5}$ inference on naive PyTorch, our scheduling optimization achieves 2.11× and 1.87× reductions in reaction time on Orin and Thor, because action prediction no longer requires invoking the VLM every time. Our graph reuse method achieves a 2.96× reduction in reaction time on Orin by eliminating per-inference graph construction overhead. Our GPU-resident intermediate buffering and flow matching unrolling achieve 1.50× and 1.59× acceleration for the action expert on the two devices, by avoiding KV communication and repeated graph invocation across multiple denoising steps. Overall, we achieve 8.66× and 3.48× improvements in control frequency on two devices, enhancing the robot's reaction speed and task performance. Compared with vla.cpp, which has a latency of 893.0ms on the Orin platform [31], Jetson-PI achieves a 5.41× improvement in control frequency through scheduling and system optimizations.

5.3 Real-world Experiments

Real-world experiments use PrimeBot Research Institute's X2-W robot, equipped with three cameras (one head camera and two wrist cameras, all at 224×224 resolution). We deploy the XR-1 model on a Jetson Orin and have the robot execute a complex task: folding clothes. This task is divided into 3 subtasks: "put the cloth in the folding area", "unfold the cloth and fold it neatly",

and "place the folded cloth in storage area". The robot executes actions at 15 Hz. As shown in Figure 7, under constrained compute resources, Jetson-PI incurs almost no accuracy loss compared to RTX 4090 deployment, and achieves a clear improvement over naive asynchronous. This shows the capability of Jetson-PI to complete complex tasks on edge devices.

5.4 Analysis of Confidence-Based Scheduling

An example of confidence evolution. Figure 9 presents an example of how confidence evolves during a single task execution. For most of the task duration, confidence remains high, and the future correction module alone is sufficient to predict environmental changes. At critical steps where environment change is hard to predict, such as during grasping and placing, confidence drops and triggers VLM invocation to ensure accurate environment information. This example demonstrates that our method adaptively balances reaction time and environmental accuracy within a single task.

Ablation on confidence threshold. Figure 8 illustrates the impact of the confidence threshold θ on robot performance. When θ is large, the VLM is invoked more frequently, leading to a slight decrease in control frequency. When θ is small, the scheduler invokes the VLM less often, which reduces reaction time but allows slightly larger environmental prediction errors, resulting in a modest drop in average confidence. Nevertheless, across different threshold values, our method achieves improvements in both success rate and control frequency compared to the configuration without scheduling optimization, demonstrating the robustness of our approach.

6 Conclusion

We presented Jetson-PI, a method for deploying VLA models on low-power onboard devices. We identified two challenges in asynchronous VLA inference: perception-execution misalignment and long reaction time. To address these, we use Foresight-Aligned Asynchronous Correction that predicts future VLM latents to guide action prediction. We use confidence-based scheduling optimization and system accelerations to reduce reaction time. Experiments on Jetson Orin show Jetson-PI achieves 8.66× and 5.41× improvements in control frequency compared with naive PyTorch and vla.cpp with negligible success rate loss.

7 Limitations

While Jetson-PI successfully enables VLA deployment on low-power onboard devices, significantly improving battery life and activity area compared to high-end GPU setups, the onboard platform remains fundamentally constrained in compute and bandwidth relative to GPU clusters. As model parameter counts and batch sizes continue to scale, the performance gains of Jetson-PI may not fully close the gap with high-end GPU deployment. Nevertheless, we believe that for mobile robotics applications where power and portability are critical, Jetson-PI offers a practical and efficient solution.

References

[1] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. pi_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

[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] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, et al. $pi\_\{0.5\}$: a vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025.

[4] M. J. Kim, C. Finn, and P. Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[5] Y. Yue, Y. Wang, B. Kang, Y. Han, S. Wang, S. Song, J. Feng, and G. Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

[6] Z. Yang, Y. Qi, T. Xie, B. Yu, S. Liu, and M. Li. Dysl-vla: Efficient vision-language-action model inference via dynamic-static layer-skipping for robot manipulation. *arXiv preprint arXiv:2602.22896*, 2026.

[7] W. Song, J. Chen, P. Ding, Y. Huang, H. Zhao, D. Wang, and H. Li. Ceed-vla: Consistency vision-language-action model with early-exit decoding. *arXiv preprint arXiv:2506.13725*, 2025.

[8] L. Su. Execution-state capsules: Graph-bound execution-state checkpoint and restore for low-latency, small-batch, on-device physical-ai serving. *arXiv preprint arXiv:2606.20537*, 2026.

[9] Z. Zhang, S. Yang, Q. Hu, L. J. Huang, J. Hou, Y. Sun, Y. Lu, and S. Han. Foreact: Steering your vla with efficient visual foresight planning. *arXiv preprint arXiv:2602.12322*, 2026.

[10] J. Tang, Y. Sun, Y. Zhao, S. Yang, Y. Lin, Z. Zhang, J. Hou, Y. Lu, Z. Liu, and S. Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

[11] J. Niu, K. Gu, Y. Zhao, S. Liang, T. Wang, X. Hu, Y. Wang, and H. Li. Realtime-vla flash: Speculative inference framework for diffusion-based vlas. *arXiv preprint arXiv:2605.13778*, 2026.

[12] C. Yang, Y. Hu, Y. Ma, Y. Yang, J. Tan, and H. Fan. Realtime-vla v2: Learning to run vlas fast, smooth, and accurate. *arXiv preprint arXiv:2603.26360*, 2026.

[13] Z. Yang, R. Chen, T. Wu, N. Wong, Y. Liang, R. Wang, R. Huang, and M. Li. Mcubert: Memory-efficient bert inference on commodity microcontrollers. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.

[14] 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.

[15] Z. Yu, B. Wang, P. Zeng, H. Zhang, J. Zhang, Z. Wang, L. Gao, J. Song, N. Sebe, and H. T. Shen. A survey on efficient vision-language-action models. *arXiv preprint arXiv:2510.24795*, 2025.

[16] W. Guan, Q. Hu, A. Li, and J. Cheng. Efficient vision-language-action models for embodied manipulation: A systematic survey. *arXiv preprint arXiv:2510.17111*, 2025.

[17] W. Jiang, J. Clemons, K. Sankaralingam, and C. Kozyrakis. How fast can i run my vla? demystifying vla inference performance with vla-perf. *arXiv preprint arXiv:2602.18397*, 2026.

[18] Z. Yang, S. Zheng, T. Xie, T. Xu, B. Yu, F. Wang, J. Tang, S. Liu, and M. Li. Efficientnav: Towards on-device object-goal navigation with navigation map caching and retrieval. *Advances in Neural Information Processing Systems*, 38:4286–4312, 2026.

[19] N. Hirose, C. Glossop, D. Shah, and S. Levine. Asyncvla: An asynchronous vla for fast and robust navigation on the edge. *arXiv preprint arXiv:2602.13476*, 2026.

[20] P. Budzianowski, W. Maa, M. Freed, J. Mo, W. Hsiao, A. Xie, T. Młoduchowski, V. Tipnis, and B. Bolte. Edgevla: Efficient vision-language-action models. *arXiv preprint arXiv:2507.14049*, 2025.

[21] 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.

[22] Y. Dai, H. Gu, T. Wang, Q. Cheng, Y. Zheng, Z. Qiu, L. Gong, W. Lou, and X. Zhou. Actionflow: A pipelined action acceleration for vision language models on edge. *arXiv preprint arXiv:2512.20276*, 2025.

[23] K. Zhang, J. Zhang, R. Xu, Y. Sun, S. Xue, Y. Wen, X. Guo, M. Guo, W. Liufu, L. Zihou, et al. A1: A fully transparent open-source, adaptive and efficient truncated vision-language-action model. *arXiv preprint arXiv:2604.05672*, 2026.

[24] Y. Li, Y. Meng, Z. Sun, K. Ji, C. Tang, J. Fan, X. Ma, S. Xia, Z. Wang, and W. Zhu. Sp-vla: A joint model scheduling and token pruning approach for vla model acceleration. *arXiv preprint arXiv:2506.12723*, 2025.

[25] R. Zhang, M. Dong, Y. Zhang, L. Heng, X. Chi, G. Dai, L. Du, D. Wang, Y. Du, and S. Zhang. Mole-vla: Dynamic layer-skipping vision language action model via mixture-of-layers for efficient robot manipulation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 18764–18772, 2026.

[26] K. Black, M. Galliker, and S. Levine. Real-time execution of action chunking flow policies. *Advances in Neural Information Processing Systems*, 38:33383–33407, 2026.

[27] K. Black, A. Z. Ren, M. Equi, and S. Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[28] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[29] Y. Zhao, L. Zhao, B. Cheng, G. Yao, X. Wen, and H. Gao. Vla-rail: A real-time asynchronous inference linker for vla models and robots. *arXiv preprint arXiv:2512.24673*, 2025.

[30] H. Wei, X. Xu, Z. Cheng, H. Yin, A. Ma, B. Yu, J. Zhou, and J. Lu. F2f-ap: Flow-to-future asynchronous policy for real-time dynamic manipulation. *arXiv preprint arXiv:2604.02408*, 2026.

[31] K. D. Nguyen, H. T. Ho, C. T. Nguyen, T. Q. Duong, L. D. Le, D. M. Nguyen, V. A. Ngo, and A. T. Le. vla. cpp: A unified inference runtime for vision-language-action models. *arXiv preprint arXiv:2606.08094*, 2026.

[32] 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*, 36:44776–44791, 2023.

[33] S. Community. Starvla: A lego-like codebase for vision-language-action model developing. *arXiv preprint arXiv:2604.05014*, 2026.

[34] J. Bjorck, F. Castañeda, N. Cherniadev, X. Da, R. Ding, L. Fan, Y. Fang, D. Fox, F. Hu, S. Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[35] Y. Chen, Y. Ge, H. Zhou, M. Ding, Y. Ge, and X. Liu. Dial: Decoupling intent and action via latent world modeling for end-to-end vla. *arXiv preprint arXiv:2603.29844*, 2026.

[36] Y. Liu, Y. Li, Z. Tang, Y. Zheng, Y. Lin, Q. Wang, Y. Li, S. Liu, S. Zhang, T. Jing, et al. Latent bridge: Feature delta prediction for efficient dual-system vision-language-action model inference. *arXiv preprint arXiv:2605.02739*, 2026.

[37] W. Yu, T. Wang, F. Li, J. Li, and L. Zhu. Acˆ 2-vla: Action-context-aware adaptive computation in vision-language-action models for efficient robotic manipulation. *arXiv preprint arXiv:2601.19634*, 2026.

[38] H. Chen, J. Liu, C. Gu, Z. Liu, R. Zhang, X. Li, X. He, Y. Guo, C.-W. Fu, S. Zhang, et al. Fast-in-slow: A dual-system foundation model unifying fast manipulation within slow reasoning. *arXiv preprint arXiv:2506.01953*, 2025.

[39] K. Sendai, M. Alvarez, T. Matsushima, Y. Matsuo, and Y. Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

[40] S. Ye, Y. Ge, K. Zheng, S. Gao, S. Yu, G. Kurian, S. Indupuru, Y. L. Tan, C. Zhu, J. Xiang, et al. World action models are zero-shot policies. *arXiv preprint arXiv:2602.15922*, 2026.

[41] T. Yuan, Z. Dong, Y. Liu, and H. Zhao. Fast-wam: Do world action models need test-time future imagination? *arXiv preprint arXiv:2603.16666*, 2026.

[42] A. Ye, B. Wang, C. Ni, G. Huang, G. Zhao, H. Li, H. Li, J. Li, J. Lv, J. Liu, et al. Gigaworld-policy: An efficient action-centered world–action model. *arXiv preprint arXiv:2603.17240*, 2026.

[43] X. Xu, H. Li, J. Ye, Y. Chen, J. Zeng, X. Chen, L. Xu, D. Lin, W. Li, and J. Pang. Futurevla: Joint visuomotor prediction for vision-language-action model. *arXiv preprint arXiv:2603.10712*, 2026.

[44] R. Zheng, J. Wang, S. Reed, J. Bjorck, Y. Fang, F. Hu, J. Jang, K. Kundalia, Z. Lin, L. Magne, et al. Flare: Robot learning with implicit world modeling. *arXiv preprint arXiv:2505.15659*, 2025.

[45] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, et al. $pi\_\{0.7\}$: a steerable generalist robotic foundation model with emergent capabilities. *arXiv preprint arXiv:2604.15483*, 2026.

[46] J. Cen, C. Yu, H. Yuan, Y. Jiang, S. Huang, J. Guo, X. Li, Y. Song, H. Luo, F. Wang, et al. Worldvla: Towards autoregressive action world model. *arXiv preprint arXiv:2506.21539*, 2025.

[47] J. Sun, W. Zhang, Z. Qi, S. Ren, Z. Liu, H. Zhu, G. Sun, X. Jin, and Z. Chen. Vla-jepa: Enhancing vision-language-action model with latent world model. *arXiv preprint arXiv:2602.10098*, 2026.

[48] Y. Su, S. Chen, H. Shi, M. Liu, Z. Zhang, N. Huang, W. Zhong, Z. Zhu, Y. Liu, and X. Liu. World guidance: World modeling in condition space for action generation. *arXiv preprint arXiv:2602.22010*, 2026.

[49] W. Zhang, H. Liu, Z. Qi, Y. Wang, X. Yu, J. Zhang, R. Dong, J. He, H. Wang, Z. Zhang, et al. Dreamvla: a vision-language-action model dreamed with comprehensive world knowledge. *Advances in Neural Information Processing Systems*, 38:24195–24228, 2026.

[50] Y. Lu, Z. Liu, X. Fan, Z. Yang, J. Hou, J. Li, K. Ding, and H. Zhao. Faster: Rethinking real-time flow vlas. *arXiv preprint arXiv:2603.19199*, 2026.

[51] Y. Shi, D. Guo, T. Zhao, F. Gao, L. Shi, C. Yu, Z. Mo, Q. Xiao, X. Peng, Q. Liao, et al. Streamingvla: Streaming vision-language-action model with action flow matching and adaptive early observation. *arXiv preprint arXiv:2603.28565*, 2026.

[52] Y. Ma, Y. Zhou, Y. Yang, T. Wang, and H. Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

[53] Q. Guo, J. Wan, S. Xu, M. Li, and Y. Wang. Hg-pipe: Vision transformer acceleration with hybrid-grained pipeline. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, pages 1–9, 2024.

[54] J. Zhang, Y. Hsieh, Z. Wan, H. Lin, X. Wang, Z. Wang, Y. Lei, and M. Zhang. Quantvla: Scale-calibrated post-training quantization for vision-language-action models. *arXiv preprint arXiv:2602.20309*, 2026.

[55] 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.

[56] S. Park, H. Kim, W. Jeon, J. Yang, B. Jeon, Y. Oh, and J. Choi. Quantization-aware imitation-learning for resource-efficient robotic control. *arXiv preprint arXiv:2412.01034*, 2024.

[57] Y. Xu, Y. Yang, Z. Fan, Y. Liu, Y. Li, B. Li, and Z. Zhang. Qvla: Not all channels are equal in vision-language-action model's quantization. *arXiv preprint arXiv:2602.03782*, 2026.

[58] H. Wang, J. Xu, Y. Xiang, J. Pan, Y. Zhou, Y.-L. Li, and G. Dai. Specprune-vla: Accelerating vision-language-action models via action-aware self-speculative pruning. *arXiv preprint arXiv:2509.05614*, 2025.

[59] T. Jiang, X. Jiang, Y. Ma, X. Wen, B. Li, K. Zhan, P. Jia, Y. Liu, S. Sun, and X. Lang. The better you learn, the smarter you prune: Towards efficient vision-language-action models via differentiable token pruning. *arXiv preprint arXiv:2509.12594*, 2025.

[60] Y. Yang, Y. Wang, Z. Wen, L. Zhongwei, C. Zou, Z. Zhang, C. Wen, and L. Zhang. Efficientvla: Training-free acceleration and compression for vision-language-action models. *Advances in Neural Information Processing Systems*, 38:40891–40914, 2026.

Table 5: Model architecture parameters of π series VLA models.

Parameter	Value
ViT (SigLIP So400m)	
ViT layers	27
ViT embedding dimension	1152
ViT patch size	14
Input Image Size	224
LLM (gemma 2b)	
LLM layers	18
LLM embedding dimension	2048
LLM attention heads	8
LLM head dimension	256
Action Expert (gemma 300m)	
Action expert layers	18
Action expert embedding dimension	1024
Action expert attention heads	8
Action expert head dimension	256
Denoising steps	10

A Model Architecture of π series Models

Table 5 presents the architectural parameters of the π series models (π_0 and $\pi_{0.5}$). The inference pipeline of these models proceeds as follows: Given raw observation (2 or 3 images) and a natural language instruction, the ViT encoder first extracts visual features, which are then projected into the token space of the LLM. The LLM processes the concatenated visual and language tokens, providing the KV cache of each layer that serves as conditioning information for the action expert. The action expert is a Diffusion Transformer (DiT) whose input is randomly initialized action noise. At each DiT layer, the AE performs cross-attention where the queries are derived from the action noise tokens, and the keys and values are formed by concatenating the KV caches from the corresponding LLM layer with the action noise tokens. Through an iterative denoising process based on flow matching, the action expert progressively refines the noise over a fixed number of steps (e.g., 10 steps during inference) to generate a smooth action chunk. This chunk contains a sequence of future actions that can be executed on the robot. Our Jetson-PI further introduces a lightweight future correction module that predicts the future VLM hidden states conditioned on the actions committed for execution, enabling the action expert to directly generate actions aligned with the future environment under asynchronous inference.

B Architecture of Future Correction Module

As illustrated in Figure 10, the future correction module takes two inputs:

- The sequence of committed actions $a_t, a_{t+1}, \ldots, a_{t+\Delta-1}$ that are guaranteed to be executed during the inference window.
- The output hidden state of the VLM final-layer at timestep t.

The module processes these inputs through a series of components. First, the action sequence is passed through an MLP and then into a Transformer block. We take the last token of its output state. Second, the VLM final-layer output at time t is compressed by a Q-Former block to produce a compact representation. These two branches are then concatenated, projected, and fed into two consecutive Transformer blocks.

From this joint representation, the module splits into two heads:

- **Correction head**: predicts the compressed VLM final-layer hidden state at time $t + \Delta t$, which is then provided to the action expert as the future environment information. In the

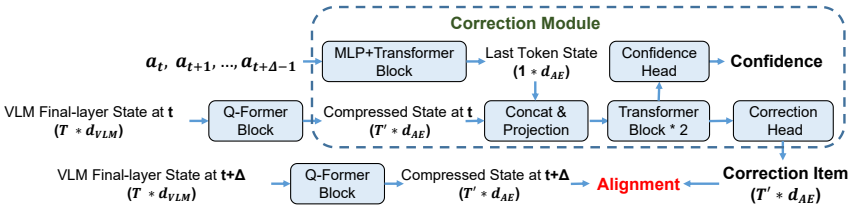

Figure 10: Architecture of future correction module. T denotes the number of tokens in the VLM hidden sequence; T' is the compressed token length after Q-Former compression (with $T' << T$, $T' = 4$ in practice); d_{VLM} is the embedding dimension of the VLM; d_{AE} is the embedding dimension of the action expert (DiT); t and $t + \Delta$ denotes time steps.

training period, the predicted correction item is aligned with the ground-truth compressed VLM final-layer hidden state at time $t + \Delta t$ as Equation 1.

- **Confidence head**: outputs a scalar confidence value $\hat{c}$ indicating the reliability of the future prediction.

The entire module is lightweight, adding minimal computational overhead, and is trained via the two-stage correction-aware training described in Section 4.1.

C Algorithm Details of Confidence-based Scheduling Optimization

Algorithm 1 shows the details of confidence-based scheduling optimization.

Algorithm 1 Confidence-based Scheduling Optimization for Asynchronous VLA Inference

Require: VLM f_{vlm}, action expert f_{act}, future correction module f_{corr}
Require: Initial observation o_0, language instruction l, confidence threshold θ ($\theta < 0.0$)
Require: Inference latency of the whole VLA model Δ, inference latency of action expert Δ_{ae}
Ensure: Continuous action execution stream

Initialize KV buffer $B_{\text{kv}} \leftarrow \emptyset$, hidden state buffer $B_h \leftarrow \emptyset$, action buffer $B_a \leftarrow \emptyset$
Compute initial VLM output: $h_0, KV_0 \leftarrow f_{\text{vlm}}(o_0, l)$
Update B_h and B_{kv}: $B_h \leftarrow h_0, B_{\text{kv}} \leftarrow KV_0$
Update actions buffer B_a: $B_a \leftarrow f_{\text{act}}(h_0, KV_0)$
Initialize parameters $\hat{c} \leftarrow 0.0,\ t \leftarrow 0$ ▷ Initialize a large confidence
while task not finished **do** ▷ Actions in B_a are continuously being executed in parallel
  if $\hat{c} > \theta$ **then** ▷ Confidence high: skip VLM, use future correction only
    Retrieve committed Δ_{ae} actions $\{a_t, a_{t+1}, \ldots, a_{t+\Delta_{ae}-1}\}$ from B_a
    Predict future VLM final-layer state: $h_{t+\Delta_{ae}} \leftarrow f_{\text{corr}}(B_h, a_t, \ldots, a_{t+\Delta_{ae}-1})$
    Obtain new confidence $\hat{c} \leftarrow f_{\text{corr}}.\text{confidence}()$
    Update actions buffer: $B_a \leftarrow f_{\text{act}}(h_{t+\Delta_{ae}}, B_{\text{kv}})$
    Update hidden state buffer: $B_h \leftarrow h_{t+\Delta_{ae}}$
  else ▷ Confidence low: invoke VLM to refresh environment context
    Observe current environment o_t
    Retrieve committed Δ actions $\{a_t, a_{t+1}, \ldots, a_{t+\Delta-1}\}$ from B_a
    Update VLM state: $h_t, KV_t \leftarrow f_{\text{vlm}}(o_t, l)$
    Update B_{kv}: $B_{\text{kv}} \leftarrow KV_t$
    Predict future VLM final-layer state: $h_{t+\Delta} \leftarrow f_{\text{corr}}(h_t, a_t, \ldots, a_{t+\Delta-1})$
    Obtain new confidence $\hat{c} \leftarrow f_{\text{corr}}.\text{confidence}()$
    Update actions buffer: $B_a \leftarrow f_{\text{act}}(h_{t+\Delta}, B_{\text{kv}})$
    Update hidden state buffer: $B_h \leftarrow h_{t+\Delta}$
  end if
  $t \leftarrow t +$ (number of executed actions)
end while
return

Table 6: Training hyperparameters for each stage of our method.

Parameter	Stage 1	Stage 2
Batch size	32	32
Learning rate	3e-4	3e-4
Optimizer	AdamW	AdamW
Number of traning steps	30,000	25,000
Learning rate schedule	Cosine	Cosine
Warmup steps	500	500
GPU type	NVIDIA A100	NVIDIA A100

D Training Details

Table 6 presents our training hyperparameters, including batch size, learning rate, optimizer, number of training steps, and so on.