How Fast Can I Run My VLA? Demystifying VLA Inference Performance with *VLA-Perf*

Wenqi Jiang
NVIDIA Research

...
NVIDIA Research

Christos Kozyrakis
NVIDIA Research

Abstract

Vision-Language-Action (VLA) models have recently demonstrated impressive capabilities across various embodied AI tasks. While deploying VLA models on real-world robots imposes strict real-time inference constraints, the inference performance landscape of VLA remains poorly understood due to the large combinatorial space of model architectures and inference systems. In this paper, we ask a fundamental research question: *How should we design future VLA models and systems to support real-time inference?* To address this question, we first introduce *VLA-Perf*, an analytical performance model that can analyze inference performance for arbitrary combinations of VLA models and inference systems. Using *VLA-Perf*, we conduct the first systematic study of the VLA inference performance landscape. From a model-design perspective, we examine how inference performance is affected by model scaling, model architectural choices, long-context video inputs, asynchronous inference, and dual-system model pipelines. From the deployment perspective, we analyze where VLA inference should be executed — on-device, on edge servers, or in the cloud — and how hardware capability and network performance jointly determine end-to-end latency. By distilling 15 key takeaways from our comprehensive evaluation, we hope this work can provide practical guidance for the design of future VLA models and inference systems.

1 Introduction

Embodied AI is widely regarded as a promising next phase of AI, with the potential to enable physical agents that can perceive, reason, and act in the real world. Notably, Vision–Language–Action (VLA) models have recently demonstrated strong capabilities in general-purpose manipulation tasks by integrating visual perception and language understanding into the action generation process [1, 2, 3, 4, 5, 6].

To react to real-time changes in the physical world, VLA inference must operate with low latency, motivating recent work to treat inference performance[1] as a first-class concern in VLA model design. Such efforts include adopting smaller models [7, 8, 9] and quantization [10, 11], skipping selected layers [12, 13], using fewer denoising steps in diffusion-based models [6], enabling asynchrony between model inference and robot execution [14, 15, 16, 17], and adopting dual-system designs comprising two models of different scales, where only the smaller model operates at high frequency [18, 19, 20].

While these efforts on efficient VLA model design are an important step forward, we still lack a comprehensive understanding of the VLA inference performance landscape, which is determined by the vast combinatorial space of possible (1) *models* and (2) *inference systems*. Here, an *inference system* is a combination of (a) the inference accelerator, ranging from edge GPUs to datacenter-class GPUs; (b) the location where inference is executed — on device, on server, or hybrid; and (c) for server-side inference, the wired or wireless network connecting the robot and the server. As we will show in our evaluation, executing the same VLA model across different inference systems can lead to performance differences of multiple orders of magnitude.

In this paper, **we present the first systematic study of VLA inference performance.** This study aims to answer a simple yet fundamental question: *how should we design VLA models and systems to achieve*

[1] In this paper, *performance* always refers to inference latency and throughput, rather than task success rate.

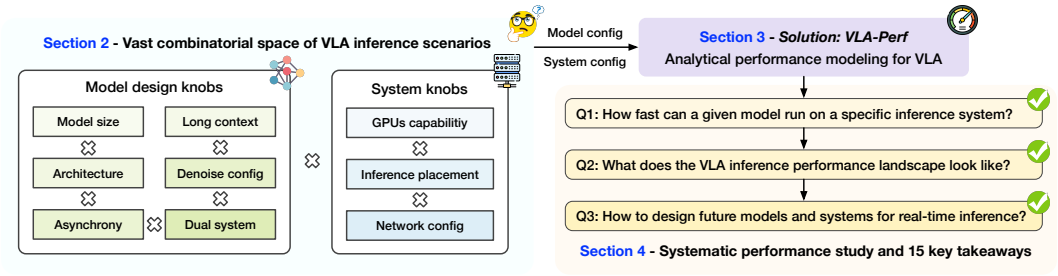


Figure 1: *VLA-Perf* enables a comprehensive performance analysis of the VLA inference landscape. Our systematic study explores the interplay between model architectures and deployment configurations, yielding 15 actionable insights for designing future VLA models and serving systems.

real-time inference performance? Given that standard RGB camera frame rates typically range from 24 to 60 Hz, we define a 10 Hz inference frequency as *acceptable* (not too far from video ingestion rates) and 100 Hz as *high-performance* (exceeding common ingestion rates). Based on this assumption, we further break down our research question to a series of concrete questions:

(1) From the perspective of VLA models, we ask: **how should future VLA models be designed under real-time performance constraints?** In particular, how much can we scale up model sizes while achieving real-time inference (§4.3)? Are long-context VLAs that possess thousands of visual frames practically feasible (§4.4)? How does the choice between autoregressive and diffusion-based action experts affect inference performance (§4.6)? How do denoising steps and action chunk size influence performance (§4.5)? How much performance gain can be achieved through asynchronous or dual-system inference (§4.9 and §4.10)?

(2) From a systems perspective, we ask: **how should we deploy efficient inference systems for various VLA workloads?** Given a model with verified accuracy, we decompose the deployment problem into the following considerations: Where should inference be executed — on device, on server, or via device–server collaboration (§4.7 and §4.8)? How to choose inference hardware given the various types of available GPUs (§4.7)? How critical is network performance in server-side inference systems (§4.7)? What combinations of models and systems are required to support VLA inference at rates from 10 Hz up to and beyond 100 Hz (§4.11)?

***VLA-Perf*.** To enable such systematic analysis across the nearly unbounded combinatorial space of VLA models and inference systems, *we develop VLA-Perf, an analytical, roofline-based performance model that predicts the optimal inference latency and throughput for arbitrary model–system combinations* (Figure 1). *VLA-Perf* supports a wide range of VLA configurations, including varying model sizes and architectures, stateless and long-context inference, different action chunk sizes, asynchronous inference, dual-system model pipelines. In addition, *VLA-Perf* supports diverse deployment scenarios, spanning inference hardware, inference locations, and network configurations. We open-source *VLA-Perf* to enable further performance analysis beyond those presented in this paper: `https://github.com/TODO/vla-perf`.
Wenqi: Internal release: https://gitlab-master.nvidia.com/wenqij/vla-perf-internal

Using *VLA-Perf*, we conduct an extensive evaluation of VLA inference performance across a broad space of model variants and system designs. From the results, we summarize **15 key performance takeaways that provide practical guidance for the design of future VLA models and inference systems.**

2 Background and Motivation

2.1 Vision-Language-Action Models

VLA models enable embodied agents to perceive the environment through vision, reason over language instructions, and generate physical actions. Recent VLA models have demonstrated strong performance on general-purpose manipulation tasks using robotic arms [21, 2, 10, 22] and humanoid robots [6, 18].

Model architecture. Existing VLA models adopt either *autoregressive-based* or *diffusion-based (including flow matching)* action generation. Autoregressive models use a single transformer to integrate visual observations, interpret language instructions, and generate actions, producing one action dimension (or token) at a time in an iterative manner. Representative examples of this paradigm include the RT

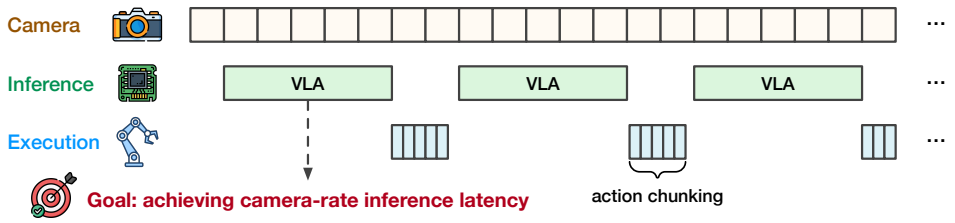


Figure 2: An example timeline of synchronous VLA inference. An efficient inference system should aim to match camera ingest rates to provide the robot with real-time action guidance.

series [21, 1, 23], OpenVLA [10], and Octo [24]. More recently, an alternative VLA paradigm combines a VLM backbone with a separate, typically smaller, diffusion-based action expert. Here, the VLM backbone ingests vision and language inputs, while a diffusion-based action expert attends to the VLM's KV cache and generates actions through an iterative refinement process, with the number of denoising steps [2] as a configurable parameter. Representative diffusion-style VLA models include the π_0 series [4, 3, 2], GR00T [6], SmolVLA [8], and TinyVLA [7].

Action prediction. Each robot action typically consists of multiple dimensions, such as joint positions, velocities, or torques for robotic arms, or whole-body joint configurations for humanoid robots. To enable smooth and stable execution, many VLA models employ *action chunking*, where the model predicts a sequence of future actions in a single inference [22, 25, 26], with the sequence length referred to as the *action chunk size*. Under action chunking, an *execution horizon* can be specified, defined as the number of actions actually executed before the next inference is performed, which is no larger than the chunk size [14, 15]. A larger execution horizon can improve action smoothness and reduce inference frequency, but it also reduces the model's ability to react promptly to changes in the external environment.

2.2 Efficient VLA Inference

An VLA system should aim to achieve real-time inference (10 to 100 ms of latency) to match the rate of visual signal ingestion, as visualized in Figure 2. To meet this demand, a growing number of work has proposed techniques to improve VLA inference efficiency at both the model and system levels [27].

Reduce computation. The amount of computation can be reduced by adopting smaller models [7, 8, 9, 28] and quantization [10, 11], skipping selected VLM layers [12, 13], or reducing the number of denoising steps in diffusion-based action experts [6]. For autoregressive VLAs, parallel decoding can further accelerate inference by reusing KV cache for multi-token predictions [25]. Finally, action chunking allows the model to predict a sequence of actions to execute in a single inference call, thereby reducing inference frequency [22, 14].

Asynchronous inference and dual-system VLAs. Inference performance can also be improved through various forms of asynchrony. For example, we can allow inference to begin while the robot is still executing previous actions [14, 15, 16, 17]. This inference-execution overlap improves GPU utilization and consequently inference throughput. Alternatively, a dual-system VLA pipeline runs a lightweight action expert at a higher frequency (System 1) while invoking a more expensive VLM backbone at a lower frequency (System 2) [18, 19, 20], with the two systems asynchronously exchanging latent states.

Better inference systems. While higher inference frequencies can be attained through more powerful hardware, software-level optimizations are also critical for VLA inference efficiency. Careful CUDA-level optimizations, including CUDA graph and operator fusion, can reduce inference latency by up to $5\times$ compared to a naive PyTorch implementation [29]. For server-side inference with action chunking, network latency and robot execution latency can be overlapped to reduce end-to-end execution time [30].

2.3 Research Gap: Comprehensive Analysis of the VLA Inference Performance Landscape

Despite the advances in efficient VLA designs introduced above, we still lack a comprehensive understanding of the VLA inference performance landscape, largely due to *(1) the wide diversity of inference system configurations across prior studies and (2) the limited exploration of model architectures driven by inference*

[2] For brevity, we use the term *denoising steps* to describe the iterative refinement steps in both classic diffusion models and flow-matching–based variants.

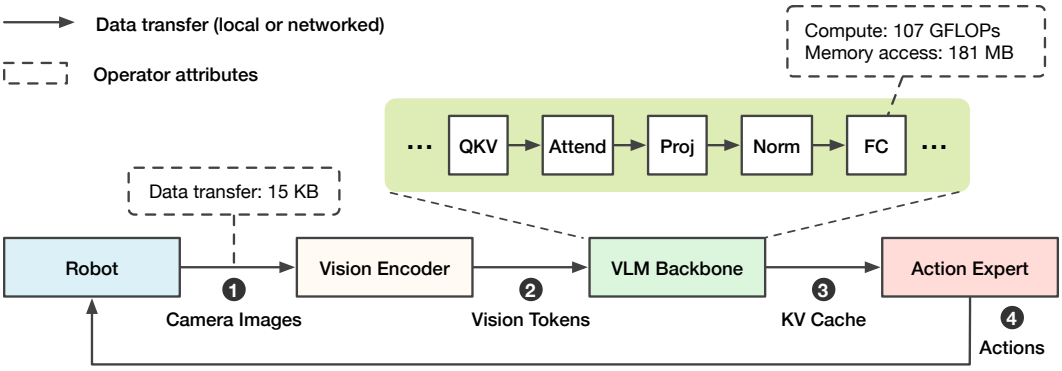


Figure 3: *VLA-Perf* abstracts VLA inference as model components interleaved with data transfers.

performance concerns. From a systems perspective, VLA inference can be performed on edge GPUs integrated within the robot (on-device) [18], on GPU servers near the robot (edge-server) [2, 4], or offloaded to powerful accelerators in the cloud (cloud-server) [21, 1] — the performance across these configurations can vary significantly, as we will show in the evaluation. From a model-design perspective, while existing VLA models are often designed with inference efficiency in mind [8, 7], they often target specific application-system pairings. Given the rapid evolution of both workloads and inference hardware, this approach can be myopic and can limit the exploration of alternative designs involving larger models or longer context.

3 Analyzing VLA Inference Performance with *VLA-Perf*

In this paper, **we aim to provide a comprehensive analysis of VLA inference performance across both existing and future, potentially hypothetical, combinations of VLA models and inference systems.** Our evaluation focuses exclusively on performance characteristics — including latency and throughput — under the assumption that the underlying models meet the necessary accuracy thresholds for deployment.

However, conducting such a comprehensive analysis is much more challenging than profiling inference performance on a small set of existing models and system implementations. This is because it requires (1) setting up systems with various accelerator capabilities, inference locations, and network configurations, as discussed in §2.3, and (2) evaluating not only existing models but also plausible future model variants — the resulting combinatorial explosion renders exhaustive empirical evaluation both cost- and time-prohibitive.

To address this challenge, **we adopt an modeling-based approach to performance analysis**, which has shown strong effectiveness in prior work on LLM inference and training systems [31, 32, 33, 34, 35, 36]. Analytical performance models focus on capturing the dominant performance characteristics of both the model (e.g., FLOPs and memory accesses) and the hardware (e.g., peak FLOP/s, memory bandwidth, and network bandwidth), and estimate achievable performance based on these attributes. This approach enables fast, low-cost performance analysis across arbitrary combinations of models and hardware, without requiring the deployment of real systems. On the downside, analytical models are not perfectly accurate, as they typically assume optimistic software implementations and therefore estimate the upper bound of achievable performance. For example, a recent study on optimizing VLA inference performance reports that 68∼75% of roofline-model-predicted performance can be achieved on real systems [29]. While such predictions are not exact, we are still at an early stage in understanding VLA inference performance, and thus even coarse-grained estimates can provide valuable guidance for future model and system designs.

***VLA-Perf* overview.** We build *VLA-Perf*, a roofline-based analytical performance model for VLA inference. Figure 3 illustrates an example VLA inference workflow, which consists of a robot and multiple model components. Depending on the placement of each model component (either on the robot or on a server), these components exchange data either locally or over a network. Such data transfers may include raw images, vision tokens, KV caches, or action predictions. Each model component is abstracted as a sequence of operators, such as fully connected layers, linear projections, and attention blocks. *VLA-Perf* assumes that inference for each individual model component (e.g., the VLM backbone) is executed on a single accelerator, because modern GPUs, including recent edge accelerators, already provide sufficient memory capacity to host complete VLA models, for example up to 128 GB on NVIDIA Jetson Thor. On the contrary, different model components can be executed either on the same accelerator or on different accelerators.

VLA Model Configuration

```
# pi0 VLM Backbone (Gemma 2B)
pi0_vlm = ModelConfig(
    seq_len=800, # language + 3 images
    hidden_size=2048,
    intermediate_size=16384,
    num_ffi=2,
    num_decoder_layers=18,
    num_attention_heads=8,
    head_dim=256,
)

# pi0 Action Expert (diffusion-based)
pi0_action_expert = ModelConfig(...)

# pi0 Vision Encoder (SigLIP)
pi0_vision_encoder = ModelConfig(...)
```

Inference System Configuration

```
# GPU Capability, e.g., NVIDIA B100
GPU_CONFIG = AcceleratorConfig(
    name='B100',
    BF16_TFLOPS=1750,
    Memory_GB'=192,
    HBM_BW_GBs=8000,
    ...
)

# Network Environment, e.g., Ethernet 1G
NET_CONFIG = NetworkConfig(
    name='Ethernet 1G',
    bandwidth_mbps=1000,
    base_latency_ms=0.1,
    efficiency=1.0
)
```

Figure 4: Example inputs to *VLA-Perf*, including model parameters (left) and system specifications (right).

Input parameters. *VLA-Perf* enables the analysis of arbitrary model-system combinations by parameterizing both model and system parameters, with an example provided in Figure 4. On the model side, these parameters include the choice of vision encoder, VLM backbone, and action expert; the input and output sequence lengths of each model; the number of denoising steps for diffusion-based action experts; action chunk size; and the dimensionality of each action. On the system side, *VLA-Perf* supports various inference accelerators of configurable peak FLOP/s and memory bandwidth, as well as network systems characterized by upload/download bandwidth and latency.

Latency calculation. Given the inputs above, the end-to-end inference latency of a VLA system is modeled as the sum of model inference latency and data movement latency across all components:

$$T_{\text{total}} = \sum_{m \in \mathcal{M}} T_m + \sum_{d \in \mathcal{D}} T_d, \tag{1}$$

where $\mathcal{M}$ denotes the set of model inference components and $\mathcal{D}$ denotes the set of data movement stages.

For a single model component m, the inference latency T_m is modeled as the sum of latency of each of its constituent operators:

$$T_m = \sum_{o \in \mathcal{O}_m} T_o, \tag{2}$$

where $\mathcal{O}_m$ denotes the sequence of operators in model m. For each operator o, *VLA-Perf* models its execution latency using a roofline model that accounts for both compute and memory access latency:

$$T_o = \max\left(\frac{\text{FLOPs}_o}{\text{FLOP/s}_h}, \frac{\text{Bytes}_o}{\text{MemBW}_h}\right), \tag{3}$$

where FLOPs_o and Bytes_o denote the total floating-point operations and memory bytes accessed by operator o, while FLOP/s_a and MemBW_a denote the peak compute throughput and memory bandwidth of the inference hardware h, respectively.

We assume that local data movement on the same accelerator is sufficiently fast to be treated as negligible, while network-based data movement between devices is modeled as:

$$T_d^{\text{net}} = \text{NetLat} + \frac{\text{Bytes}_d}{\text{NetBW}}, \tag{4}$$

where Bytes_d denotes the amount of transferred data, and NetBW and NetLat denote the single-directional network bandwidth and latency, respectively.

Modeling accuracy. Due to the scarcity of well-optimized frameworks for VLA inference, we mainly validate the accuracy of *VLA-Perf* using the π_0 implementation by Ma et al. [29], a Triton-based implementation specifically tuned for RTX 4090. Table 1 compares the performance predicted by *VLA-Perf* to the empirical measurements conducted by Ma et al. [29]. The results demonstrate that an optimized system can achieve $73.3 \sim 82.6\%$ of the theoretical roofline reported by *VLA-Perf*, with the gap narrowing as the workload increases (e.g., when processing three camera frames).

The performance differences between a real inference system and the roofline limits reported by *VLA-Perf* are due to both hardware and software factors. First, our model abstracts away hardware-specific details,

Table 1: Roofline model validation against real π_0 Triton inference latencies on an RTX 4090 [29]. This evaluation uses 10 flow-matching steps, action chunk size of 63, and an empty language prompt.

Metric	1 camera	2 cameras	3 cameras
Roofline (*VLA-Perf*)	14.7 ms	22.5 ms	30.4 ms
Real Perf. (Triton)	20.0 ms	27.3 ms	36.8 ms
VLA-Perf Accuracy	73.3%	82.3%	82.6%

including microarchitectural design, instruction scheduling, and memory-access behavior. Instead, *VLA-Perf* assumes that the maximum theoretical compute capability and memory bandwidth are attainable for every operator executed. Second, real-world systems incur software overheads, such as kernel launch latencies, operating system interference, and runtime library overhead, which are not explicitly modeled by *VLA-Perf*. Nevertheless, we believe that a modeling accuracy exceeding 80% is sufficient to provide meaningful insights into the VLA performance landscape, and thus leave the tuning of *VLA-Perf* for specific hardware and software platforms to future work.

4 Evaluation and Takeaways

In this section, we use *VLA-Perf* to conduct a comprehensive analysis of VLA inference performance. Our evaluation is structured to address two sets of research questions as below.

Question 1: How should we design future VLA models to meet real-time latency constraints?

- How far can model sizes be scaled while still enabling real-time inference (§4.3)?
- Are long-context VLAs that process thousands of visual frames practically feasible (§4.4)?
- How do autoregressive and diffusion-based action experts compare in performance (§4.6)?
- How do denoising steps and action chunk size influence performance (§4.5)?
- Are asynchronous or dual-system inference much faster than synchronous inference (§4.9 and §4.10)?

Question 2: How should inference systems be deployed for different VLA workloads?

- Should inference be executed on device, on server, or via device-server collaboration (§4.7 and §4.8)?
- How capable must inference hardware be to meet real-time performance requirements (§4.7)?
- How critical is network performance for server-side inference systems (§4.7)?
- What model-system combinations can achieve inference rates from 10 Hz to 100 Hz (§4.11)?

Our experiments are organized as follows. §4.2 presents a baseline analysis of the π_0 model. §4.3∼4.6 explore various model configuration to examine their impact on inference performance. §4.7∼4.11 additionally considers inference placement and network latency, closely reflecting real-world deployments.

4.1 Evaluation Setup

We describe the main model and system settings here, with additional details provided in Appendix A.

Models and robot. We evaluate a set of model variants derived from the π_0 architecture [2], which we choose due to its strong robotic task performance and widespread adoption in recent VLA systems. The original π_0 model consists of a 400M SigLIP vision encoder, a 2B Gemma language model, and a 300M diffusion-based action expert. Throughout the experiments, we consider on a bimanual robotic manipulation setting, which is common for both stationary robots and mobile platforms such as wheeled or humanoid robots. With the UR5e robot arm, this setup is equipped with three cameras and an action space of 14 degrees of freedom (DoF). Each camera image has a resolution of 224×224 and is tokenized into 256 visual tokens, yielding 768 visual tokens across three cameras. Assuming 32 language tokens per task, the total input sequence length per inference is 800 tokens. Unless otherwise specified, we use an action chunk size of 50 and 10 denoising steps for action generation, same as the original π_0 configuration.

Inference systems. We evaluate a range of accelerators spanning high-end edge GPUs (e.g., NVIDIA Jetson Thor), consumer-grade GPUs commonly used in research experiments (e.g., RTX 4090), and high-end datacenter GPUs, including A100, H100, and B100. The GPUs can be mapped to various

Table 2: We consider systems with various (1) GPU capabilities (rows) and (2) inference location (columns).

Capacity and Placement	On-Device	Edge Server	Cloud Server
Mobile (Thor)	✓		
Consumer (RTX 4090)		✓	
Datacenter (B100)		✓	✓

Table 3: Inference performance of π_0 on various GPUs without considering network latency.

Hardware	Vision Lat.	VLM Lat.	Action Lat.	E2E Lat.	E2E Freq.
Jetson Thor	6.06 ms	20.30 ms	26.20 ms	52.57 ms	19.0 Hz
RTX 4090	4.02 ms	19.79 ms	7.25 ms	31.06 ms	32.2 Hz
A100	2.13 ms	10.47 ms	3.60 ms	16.20 ms	61.7 Hz
H100	0.71 ms	3.30 ms	2.14 ms	6.15 ms	162.5 Hz
B100	0.40 ms	1.87 ms	0.91 ms	3.18 ms	314.4 Hz

Table 4: Compute- vs. memory-bound analysis of π_0 across different hardware. Operator intensity (OI) denotes the ratio between compute operations and memory accesses (FLOPs/Bytes). The balance OI denotes the hardware balance point at which compute throughput and memory bandwidth are equally limiting.

Hardware	Balance OI	**Vision** (OI=321.4)	**VLM** (OI=542.8)	**Action** (OI=54.0)
Jetson Thor	1481.5	Memory	Memory	Memory
RTX 4090	163.7	Compute	Compute	Memory
A100	153.0	Compute	Compute	Memory
H100	295.2	Compute	Compute	Memory
B100	218.8	Compute	Compute	Memory

inference location as shown in Table 2. For server-side inference, we evaluate both wired and wireless network configurations, including Ethernet, WiFi, and cellular (4G/5G) networks. All experiments assume BF16 for inference, or FP16 when BF16 is not supported by the hardware.

Performance metrics. We report VLA system latency, defined as the elapsed time from when the robot perceives visual observations to when the robot receives the corresponding action prediction. We also report throughput in Hertz (Hz), defined as the number of inference that can be executed per second at a batch size of one (i.e., a single robot). For synchronous inference, throughput is the inverse of inference latency, whereas for asynchronous inference, throughput can exceed the inverse latency. We report inference performance independent of robot execution latency, as the latter is highly robot-dependent. Furthermore, the effective action execution frequency may exceed the inference frequency due to action chunking — with a chunk size of five, the robot can execute actions at up to five actions given a single inference.

4.2 Baseline π_0 Inference Latency Across Hardware Backends

Before evaluating model and system variants, we first establish a baseline by measuring the inference performance of the π_0 model across a range of GPUs, without considering network latency.

Takeaway 1: Existing datacenter GPUs can already achieve inference frequencies comparable to camera frame rates for small VLA models such as π_0, while edge GPUs remain performance-limited.

Table 3 shows that A100, H100, and B100 achieve inference frequencies ranging from 61.7 Hz to 314.4 Hz, which are at least on par with the frame rates of common RGB cameras (24∼60 Hz). In contrast, Jetson Thor achieve substantially lower inference frequency (19.0 Hz), falling below the frame rates of most cameras.

Takeaway 2: Action prediction is memory-bound across hardware, while vision and VLM inference are compute-bound on most GPUs except from Jetson Thor.

Table 4 summarizes the workload characteristics of each VLA model component. The vision encoder and the VLM backbone exhibit significantly higher operator intensity (321.4 and 542.8 FLOPs/Byte, respectively) compared to the action expert (54.0 FLOPs/Byte). This is because the vision encoder and

Table 5: Inference performance of scaled-up VLA models across different hardware platforms.

Model	Vision Encoder	VLM	Action Expert	Jetson Thor	RTX 4090	B100
π_0 (2.7B)	SigLIP-So (0.4B)	Gemma-2B (2.0B)	Act-M (0.3B)	19.0 Hz	32.2 Hz	314.4 Hz
π_0-L (9.1B)	SigLIP-Giant (1.1B)	Llama2-7B (6.5B)	Act-L (1.5B)	3.9 Hz	8.0 Hz	73.6 Hz
π_0-XL (16.7B)	SigLIP-Giant (1.1B)	Llama2-13B (12.7B)	Act-XL (2.9B)	2.1 Hz	N/A	39.7 Hz
π_0-XXL (81.3B)	SigLIP-Giant (1.1B)	Llama2-70B (68.5B)	Act-XXL (11.7B)	N/A	N/A	9.6 Hz

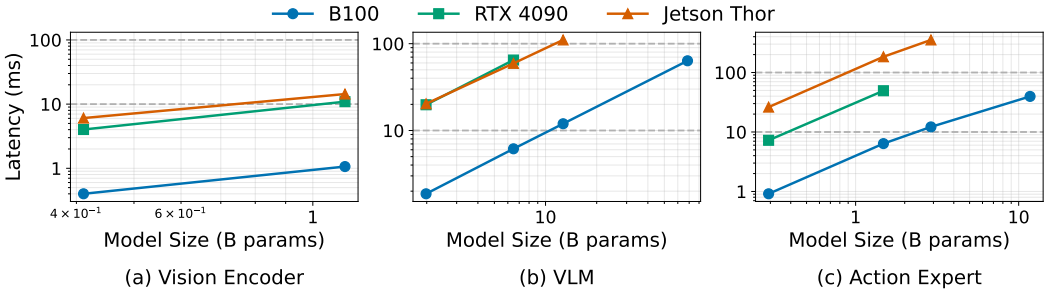


Figure 5: Increased model sizes lead to proportional inference latency increases.

the VLM backbone process many input tokens (e.g., 768 for SigLIP and 800 for Gemma), inherently batching computation across tokens, whereas the diffusion-based action expert operates on far fewer tokens (e.g., the same as the action chunk size of 50). This behavior closely mirrors LLM inference, where the prefill phase (prompt processing) is compute-intensive, while the decode phase (token generation) is memory-bound [37]. Jetson Thor, in contrast to the other evaluated GPUs, relies on LPDDR memory, which prioritizes low power consumption for embedded devices but provides substantially lower bandwidth (270 GB/s) than GDDR on RTX 4090 (1 TB/s) and HBM on B100 (8 TB/s). As a result, even the vision encoder and VLM backbone become memory-bound on Jetson Thor.

4.3 Scaling Model Sizes Under Real-Time Constraints

We next study how inference latency scales with increasing VLA model sizes, which are positively correlated with task accuracy [38]. Specifically, we scale each component of the π_0 model and construct a family of larger VLA models. For the vision encoder, we replace the original SigLIP-So400m used in π_0 with the larger SigLIP-Giant model with 1.1B parameters. For the VLM, we replace Gemma with the Llama2 family (7B, 13B, and 70B), which provides a wider range of model scales. For the action expert, we follow the π_0 design principle and instantiate it as a scaled-down version of the corresponding VLM, with approximately $4\sim8\times$ fewer parameters by reducing the transformer hidden dimension and intermediate dimension by $2\times$ and $4\times$, respectively. By combining these components, we construct a set of hypothetical larger VLA models, denoted as π_0-L, π_0-XL, and π_0-XXL, whose configurations are summarized in Table 5.

Takeaway 3: Latency of each VLA component scales approximately linearly with increasing model sizes.

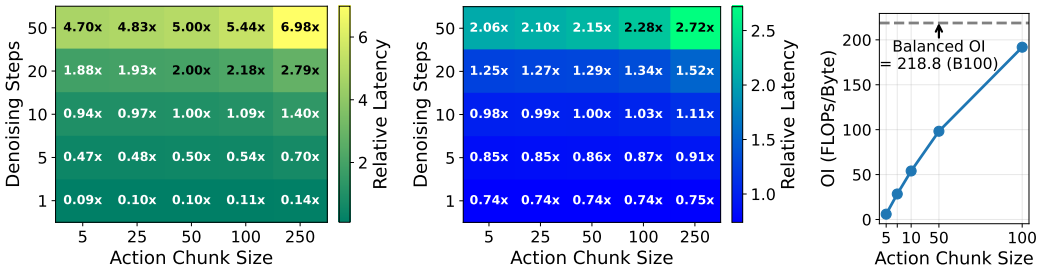
Figure 5 breaks down the inference latency of individual VLA components as model size increases, where both axes are shown on a logarithmic scale. Across all components, larger models impose proportionally higher computational costs, and thus inference latency grows approximately linearly with model size.

Takeaway 4: While edge and consumer GPUs struggle with larger models, datacenter GPUs can still support real-time inference for VLA models that are more than one order of magnitude larger.

Table 5 summarizes inference performance across different model scales. B100 sustains 9.6 Hz inference even for the largest 81B model variant ($30\times$ larger than π_0), demonstrating that modern datacenter GPUs can accommodate substantially larger VLA models under real-time constraints. In contrast, RTX 4090 runs out of memory for π_0-XL (16.7B), and Jetson Thor struggles to deliver real-time performance even with sufficient memory capacity, achieving only 2.1 Hz inference frequency on π_0-XL.

Table 6: Inference performance and memory consumption of long-context VLA models.

Timesteps	Total Memory	KV Cache Size	Jetson Thor	RTX 4090	B100
1	5.1 GB	0.01 GB	52.6 ms (19.0 Hz)	31.1 ms (32.2 Hz)	3.2 ms (314.4 Hz)
10	5.3 GB	0.13 GB	58.4 ms (17.1 Hz)	39.0 ms (25.7 Hz)	3.9 ms (254.6 Hz)
100	6.4 GB	1.3 GB	122.9 ms (8.1 Hz)	117.3 ms (8.5 Hz)	11.3 ms (88.4 Hz)
1000	18.3 GB	13.2 GB	768.3 ms (1.3 Hz)	900.6 ms (1.1 Hz)	85.2 ms (11.7 Hz)
10000	137.0 GB	131.8 GB	N/A	N/A	823.7 ms (1.2 Hz)



(a) Action Expert Latency. (b) VLA Total Latency. (c) Action Expert OI.

Figure 6: The impact of denoising steps and action chunk sizes to inference performance on B100 GPU.

4.4 Long-Context VLA Inference

While π_0 model ipredicts actions solely based on the current observation, this a memory-less design is insufficient for long-horizon tasks that require reasoning over temporal context [39, 40, 24, 41]. In this section, we adapt π_0 to a stateful setting by enabling it to incorporate past visual states into the VLM KV cache. At each new timestep, the latest visual inputs (three camera images, corresponding to 768 vision tokens) attend over the accumulated KV cache of the VLM, and the action prediction is conditioned on this long context.

Takeaway 5: Datacenter GPUs can support real-time long-context VLA inference with up to 1K past timesteps, while edge and consumer GPUs are limited to roughly 100 steps.

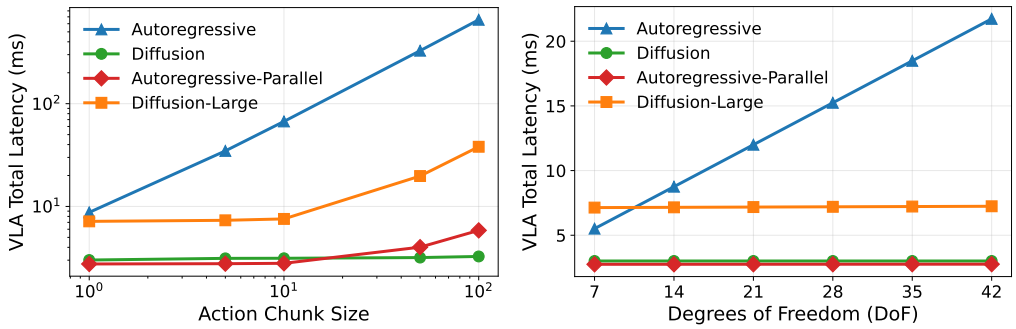
Table 6 reports inference performance and memory consumption of long-context VLA with up to 10K past timesteps. B100 sustains 11.7 Hz inference with 1K past timesteps, whereas performance drops to 1.2 Hz at 10K steps, which no longer meets real-time requirements. For Jetson Thor and RTX 4090, real-time performance is only achievable when the context length is limited to roughly 100 timesteps (around 8 Hz).

4.5 Impact of Denoising Steps and Action Chunk Size

Given a diffusion-based action expert model, two key parameters influence inference performance: (1) the number of denoising steps, where each step incurs a forward pass, and (2) the action chunk size, i.e., the number of predicted actions. To this end, we vary the number of diffusion steps of π_0 from 1 to 50 (default: 10) and the action chunk size from 5 to 250 (default: 50), each spanning a $50\times$ range. For brevity, we present results on B100, but the observed trends below are consistent across all evaluated GPUs.

Takeaway 6: Denoising steps have a significant impact on both action expert latency and end-to-end VLA latency, whereas action chunk size has a negligible effect.

Figure 6a and Figure 6b report the action expert latency and end-to-end VLA inference latency, respectively. On the one hand, action prediction latency scales linearly with the number of diffusion steps and thus has a substantial impact on overall VLA latency. For example, with the default action chunk size of 50, increasing the number of diffusion steps from 10 to 50 leads to a proportional increase in action prediction latency ($5\times$) and a $2.15\times$ increase in overall VLA latency. On the other hand, action chunk size has only a marginal effect on both action-expert latency and end-to-end VLA inference latency. With the default setting of 10 denoising steps, increasing the action chunk size from 50 to 250 ($5\times$) increases action prediction latency by only 40%, resulting in just an 11% increase in end-to-end VLA latency. This is because action prediction is typically memory-bound (Figure 6c): performance is limited by loading model parameters and KV cache from memory, and the additional computation given more action tokens has little effect on overall latency.



(a) Varying action chunk sizes.

(b) Varying DoF (no action chunking).

Figure 7: Diffusion vs autoregressive VLA inference performance on B100 GPU.

4.6 Diffusion-Based vs. Autoregressive Action Prediction

Diffusion-based and autoregressive action prediction are two dominant paradigms in recent VLA models. Autoregressive action decoder typically uses the same transformer to process vision and language inputs and to generate actions [10, 24]. Accordingly, we adapt π_0 to an autoregressive variant that uses the VLM backbone directly for action prediction. In contrast, diffusion-based VLAs usually employ a separate action expert that is significantly smaller than the VLM (e.g., $6.7\times$ smaller in π_0) [2, 6]. For fairness, we additionally evaluate a diffusion-based variant with an action expert that matches the VLM size, referred to as *Diffusion-Large*. Classic autoregressive VLAs generate one action dimension at a time, which results in high inference latency as it requires many sequential prediction steps (e.g., 700 steps in our case with a 14-DoF action space and an action chunk size of 50). Thus, we also evaluate a faster autoregressive variant with parallel decoding [25], which predicts all actions in a single inference, denoted as *Autoregressive-Parallel*.

Takeaway 7: With action chunking, diffusion-based VLA inference is one to two orders of magnitude faster than the vanilla autoregressive VLA.

Figure 7a compares inference latency across different architectures on the B100 GPU. Across all action chunk sizes, diffusion-based models (both the standard and large variants) consistently outperform the vanilla autoregressive VLA. With the default chunk size of 50, the standard diffusion model achieves an inference latency of 3.2 ms, which is $102.4\times$ faster than the classic autoregressive model (327.6 ms).

Takeaway 8: Autoregressive VLAs are competitive only when generating a small number of action tokens or when parallel decoding is enabled.

To further analyze scenarios where autoregressive VLAs can be efficient, we evaluate inference performance without action chunking across common action dimensionalities, ranging from 7 DoF for a single robot arm [2, 1] to over 40 DoF for two dexterous hands [42, 43]. As shown in Figure 7b, the autoregressive model can slightly outperform the large diffusion-based model when the number of generated action tokens is small (e.g., 7), although the standard-size diffusion model remains faster. Another scenario in which autoregressive inference becomes competitive is when parallel decoding is employed. As shown in Figure 7a, parallel decoding outperforms the standard diffusion model for action chunk sizes up to 10. However, for larger chunk sizes such as 50, the latency of parallel decoding increases substantially as the workload transitions from memory-bound to compute-bound (OI increases from 135.9 at chunk size 10 to 477.7 at chunk size 50, exceeding the B100 balance OI of 218.8). In contrast, the diffusion-based action expert remains memory-bound (OI = 54.0 at chunk size of 50), leading to more stable inference performance across chunk sizes.

4.7 On-Device vs. Server-Side Inference

We evaluate three classes of VLA inference systems with different GPU and network configurations. First, *on-device inference*, where inference is executed directly on an edge GPU integrated into the robot (e.g., Jetson Thor), as demonstrated by systems such as Figure AI's Helix [18]. Second, *edge-server inference*, where inference is performed on a server located close to the robot [2, 6, 30]. In this setting, communication between the robot and the server may use wired networks (Ethernet) for fixed-base robots or wireless networks (WiFi or cellular networks) for mobile robots (e.g., wheeled or humanoid robots), while the server-side accelerator may range from consumer-grade GPUs (e.g., RTX 4090) to datacenter-class GPUs

Table 7: Network configuration specifications.

Metric	Ethernet 1G	Ethernet 10G	WiFi 6	WiFi 7	4G	5G	Slow Cloud	Fast Cloud
Upload BW	1 Gbps	10 Gbps	560 Mbps	2 Gbps	19 Mbps	80 Mbps	1 Gbps	10 Gbps
Download BW	1 Gbps	10 Gbps	800 Mbps	3 Gbps	75 Mbps	500 Mbps	1 Gbps	10 Gbps
Base Latency	0.10 ms	0.05 ms	3.50 ms	2.50 ms	25.00 ms	10.00 ms	100.00 ms	10.00 ms

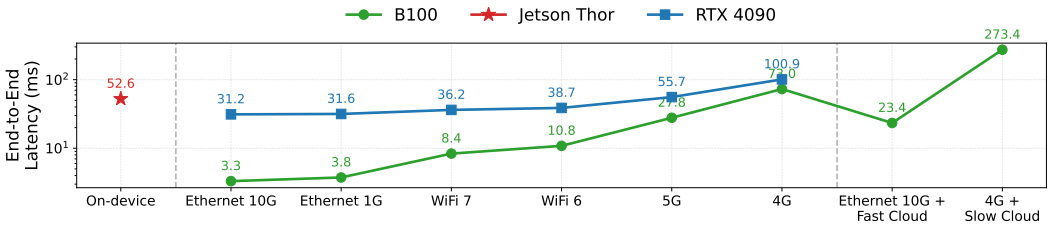


Figure 8: Inference performance on device, on edge servers, and on cloud servers.

(e.g., B100). Third, *cloud-server inference*, where inference runs on high-end datacenter GPUs. In this case, the robot first communicates with a nearby gateway server via a wired or wireless connection, which then forwards inference requests to the cloud, incurring two stages of communication latency. Note that network latency to cloud servers can vary substantially, depending on factors such as physical distance and routing topology [44, 45]; thus, we consider two cloud network configurations with different performance. We summarize detailed network performance parameters in Table 7.

Takeaway 9: Server-side inference, even with only consumer GPUs, significantly outperforms on-device inference in most scenarios, except under extremely poor network conditions.

As shown in Figure 8, even when using a consumer GPU (RTX 4090) connected via WiFi, server-side inference achieves lower end-to-end latency than on-device inference on Jetson Thor. With a more powerful B100 GPU, inference remains faster than on-device execution even when deployed on an edge server with only cellular (5G) connectivity or in a cloud instance with fast network. On-device inference becomes preferable only when network conditions are extremely constrained, such as (i) slow cellular connections (4G or below), or (ii) cloud deployments where the datacenter is distant from the robot.

4.8 Device-Server Collaborative Inference

Some robots already have an on-board GPU — so a natural idea is to split the VLA workload between server and device to (1) reduce server workloads and to (2) improve performance over device-only deployments. Since the action expert model is usually several times smaller than the VLM backbone [2, 8, 7], a natural idea is to run VLM inference (including the vision encoder) on server (B100) and run action expert inference on device (Jetson Thor). In comparison to either device-only or server-only solutions, here, device-server collaboration involves an extra communication step, where the KV cache of the VLM has to be downloaded to the device GPU before action prediction begins.

Takeaway 10. Device–server collaboration is often slower than device-only inference and always slower than server-side inference, making this solution generally unattractive in practice.

As shown in Figure 9, collaborative inference is always slower compared to server-only inference — which is not surprising as now the action expert runs on a less powerful device. What we found interesting is that it is even slower than on-device inference in most cases, except with a fast wired network (Ethernet 10G) — this is because of the KV cache download process from the server to the device, which can be very slow without a fast network (12.4, 43.7, and 257.7 ms for Ethernet 10G, WiFI 7, and 5G networks, respectively). However, we argue that such scenarios are rare in practice: robots equipped with on-device GPUs are typically mobile platforms that relies on wireless connectivity, in which case using the on-device GPU alone (rather than device-server collaboration) is the more performant choice.

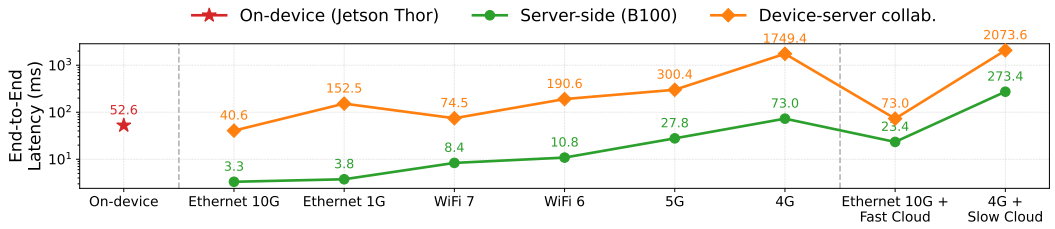


Figure 9: Device-server collaborative inference versus server-only and device-only solutions.

Table 8: Inference frequency of synchronous and asynchronous systems.

Hardware	Network	Latency	Freq. (Sync)	Freq. (Async)	Speedup
B100	Ethernet 10G	3.3 ms	301.4 Hz	314.4 Hz	1.04×
B100	Ethernet 1G	3.8 ms	266.5 Hz	314.4 Hz	1.18×
B100	WiFi 7	8.4 ms	119.7 Hz	314.4 Hz	2.63×
B100	5G	27.8 ms	35.9 Hz	215.3 Hz	5.99×
B100	4G	73.0 ms	13.7 Hz	50.5 Hz	3.68×
B100	Wired + Fast Cloud	23.4 ms	42.8 Hz	314.4 Hz	7.34×
B100	4G + Slow Cloud	273.4 ms	3.7 Hz	50.5 Hz	13.79×

4.9 Asynchronous Inference

With asynchronous inference, the model predicts actions based on stale observations rather than the latest state, allowing model inference and robot action execution to be partially overlapped. While this form of asynchrony does not increase the maximum inference throughput for on-device inference without network latency, it can benefit server-side inference by allowing network transmission and GPU computation to proceed concurrently. Thus, the asynchronous inference throughput is bounded by the minimum of GPU inference throughput and network transmission throughput.

Takeaway 11: Asynchrony between robot execution and inference can significantly improve system throughput for server-side inference, especially under slow wireless network connections.

Table 8 reports the server-side inference throughput under different network configurations. With fast wired networks (e.g., 1 GbE and 10 GbE Ethernet), synchronous and asynchronous inference achieve similar throughput. In contrast, under slower wireless networks (WiFi 7, 5G, and 4G), asynchronous inference improves throughput by 2.63∼5.99×. With WiFi 7, inference remains GPU-bound, and thus achieves the same throughput to wired networks (314.4Hz). For 5G and 4G, the bottleneck shifts to network transmission, resulting in lower asynchronous throughput. Note that while asynchronous inference improves throughput, it does not reduce end-to-end latency; increased staleness may degrade action quality, which warrants further investigation from the perspective of control stability and task success rate.

4.10 Dual-system VLA Pipelines

Recent work proposes a *System 1 + System 2* paradigm for action generation, where a slower System 2 (the VLM) responsible for high-level reasoning operates at a lower frequency (e.g., 5–10 Hz), while a faster System 1 (the action model) reacts to the environment at a higher frequency using the most recent visual inputs [18, 19, 20]. The two systems run asynchronously: the action expert conditions its predictions on the VLM's KV cache, which is updated at a lower frequency by System 2. While this design is conceptually appealing, we are not aware of a widely adopted, open-source diffusion-style implementation of a dual-system VLA. Therefore, we make the following approximations in our evaluation: (1) System 1 latency consists of image upload, vision encoding, diffusion-based action prediction, and action download, where the cost of integrating vision features into the action expert is assumed to be negligible; and (2) System 2 latency equals VLM inference, which incorporates the visual encoding of the most recently uploaded image.

Takeaway 12: Asynchronous inference between System 1 and System 2 can improve action prediction performance, with performance gains strongly dependent on hardware capability and network latency.

Table 9: Performance gains by using dual-system inference.

Hardware	Network	S1 Lat.	S2 Lat.	Freq. (Sync)	S2 Cap = 5 Hz		S2 Cap = 10 Hz	
					Freq. (Async)	Speedup	Freq. (Async)	Speedup
Jetson Thor	On-device	32.3 ms	20.3 ms	19.0 Hz	27.8 Hz	1.46×	24.7 Hz	1.30×
B100	Ethernet 10G	1.5 ms	1.9 ms	301.4 Hz	682.4 Hz	2.26×	676.0 Hz	2.24×
B100	WiFi 7	6.5 ms	1.9 ms	119.7 Hz	152.6 Hz	1.28×	151.2 Hz	1.26×
B100	5G	26.0 ms	1.9 ms	35.9 Hz	38.2 Hz	1.06×	37.8 Hz	1.05×

Table 9 reports performance across different system configurations and System 2 frequency caps (5 Hz and 10 Hz). On Jetson Thor, the improvement is moderate (1.46× at a 5 Hz cap and 1.30× at a 10 Hz cap), since the asynchronous frequency cap is comparable to the synchronous VLM frequency of 19 Hz. In contrast, on B100 with a fast 10G Ethernet connection, the speedup is substantial (2.24× at a 10 Hz cap). In this case, asynchronous execution significantly reduces the effective VLM invocation rate from 301.4 Hz to 10 Hz, freeing compute resources that can be reallocated to action prediction. However, under slower network conditions (e.g., 5G), the benefit diminishes, yielding only a 1.05× speedup at a 10 Hz cap. This is because network latency substantially increases System 1 latency — from 1.5 ms with Ethernet to 26.0 ms with 5G — thereby limiting the achievable performance regardless of the inference hardware capability.

4.11 Supporting High-Performance VLA Inference up to 100 Hz

In this section, we analyze how 10 Hz and 100 Hz performance targets (§2) can be achieved with the π_0 model across on-device, edge-server, and cloud-server inference systems. We also discuss what algorithm-level adjustments may be required when the target performance cannot be met by those systems.

Takeaway 13: For on-device inference, the most advanced edge GPUs (Jetson Thor) can already achieve 10 Hz inference for π_0, but reaching 100 Hz requires model-level adjustments.

Table 3 shows that Jetson Thor already achieves 19 Hz inference throughput for π_0, exceeding the 10 Hz target. However, achieving 100 Hz would require roughly a 5× improvement. This gap have to be closed through model-level optimizations, such as reducing model size (§4.3), decreasing the number of diffusion steps (§4.5), or using lower-precision quantization [10, 11].

Takeaway 14: For edge-server inference, 10 Hz is achievable with consumer GPUs and wireless networks, while 100 Hz requires datacenter GPUs and faster networks.

Figure 8 shows that an RTX 4090 can achieve 10 Hz inference even with a slow 4G network. However, achieving sub-10 ms latency (100 Hz) requires either a more powerful accelerator such as B100 or the aforementioned model-level optimizations. For B100, reaching 100 Hz further depends on network performance, requiring either wired Ethernet or high-quality wireless connectivity (e.g., WiFi 7).

Takeaway 15: For cloud-server inference, 10 Hz is feasible with good networking, while achieving 100 Hz generally requires asynchronous inference.

As shown in Table 8, B100 achieves only 42.8 Hz under synchronous cloud inference even with a fast network. In this regime, network latency alone (exceeding 10 ms per upload or download) prevents achieving 100 Hz, making computation reduction insufficient and rendering asynchronous inference necessary for high-frequency operation. With a fast network (WiFi 7 or better), asynchronous execution cab achieve a throughput of 314.4 Hz. Even under poor network conditions where synchronous inference becomes unacceptable (3.7 Hz), asynchronous inference can still restore acceptable performance (50.5 Hz).

5 Conclusion and Future Work

We present the first comprehensive study of VLA inference performance. Using *VLA-Perf*, an analytical performance modeling tool that we develop, we systematically explore a wide range of (1) model configurations, including model size, context length, architectural choices, and synchronous versus asynchronous execution, and (2) system configurations spanning different hardware platforms, inference placements, and network conditions. From the performance study, we distill 15 key takeaways that provide practical guidance for the design of future VLA models and inference systems.

While this work represents an important step toward understanding and building next-generation VLA systems, we view it as only a starting point. First, our study focuses primarily on VLA models for manipulation tasks and does not consider other embodied AI domains such as autonomous driving, quadrupeds, or drones. These settings often involve different system constraints (e.g., stronger emphasis on on-device execution) and additional model components (e.g., SLAM and specialized control modules), which are beyond the scope of this work. Second, robotic systems are complex end-to-end pipelines that go beyond model inference alone. In this work, we do not account for robot execution latency or sensor latency (e.g., cameras), as these factors vary widely across platforms. A more comprehensive performance analysis that integrates inference, sensing, and actuation would provide deeper insights into end-to-end robotic system behavior. We leave these directions to future work.

References

[1] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, et al. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*, pages 2165–2183. PMLR, 2023.

[2] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, et al. π0: A visionlanguage-action flow model for general robot control, 2024a. *URL https://arxiv. org/abs/2410.24164*, 2024.

[3] P Intelligence, K Black, N Brown, J Darpinian, K Dhabalia, D Driess, A Esmail, M Equi, C Finn, N Fusai, et al. π0.5: A vision-language-action model with open-world generalization. arxiv 2025. *arXiv preprint arXiv:2504.16054*.

[4] Ali Amin, Raichelle Aniceto, Ashwin Balakrishna, Kevin Black, Ken Conley, Grace Connors, James Darpinian, Karan Dhabalia, Jared DiCarlo, Danny Driess, et al. π0.6: a vla that learns from experience. *arXiv preprint arXiv:2511.14759*, 2025.

[5] Gemini Robotics Team, Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

[6] Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

[7] Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Zhibin Tang, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, et al. Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025.

[8] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, et al. Smolvla: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025.

[9] Tao Lin, Yilei Zhong, Yuxin Du, Jingjing Zhang, Jiting Liu, Yinxinyu Chen, Encheng Gu, Ziyan Liu, Hongyi Cai, Yanwen Zou, et al. Evo-1: Lightweight vision-language-action model with preserved semantic alignment. *arXiv preprint arXiv:2511.04555*, 2025.

[10] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan Foster, Grace Lam, Pannag Sanketi, et al. Openvla: An open-source vision-language-action model. *arXiv preprint arXiv:2406.09246*, 2024.

[11] Hongyu Wang, Chuyan Xiong, Ruiping Wang, and Xilin Chen. Bitvla: 1-bit vision-language-action models for robotics manipulation. *arXiv preprint arXiv:2506.07530*, 2025.

[12] Yang Yue, Yulin Wang, Bingyi Kang, Yizeng Han, Shenzhi Wang, Shiji Song, Jiashi Feng, and Gao Huang. Deer-vla: Dynamic inference of multimodal large language models for efficient robot execution. *Advances in Neural Information Processing Systems*, 37:56619–56643, 2024.

[13] Zebin Yang, Yijiahao Qi, Tong Xie, Bo Yu, Shaoshan Liu, and Meng Li. Dysl-vla: Efficient vision-language-action model inference via dynamic-static layer-skipping for robot manipulation.

[14] Kevin Black, Manuel Y Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. *arXiv preprint arXiv:2506.07339*, 2025.

[15] Kevin Black, Allen Z Ren, Michael Equi, and Sergey Levine. Training-time action conditioning for efficient real-time chunking. *arXiv preprint arXiv:2512.05964*, 2025.

[16] Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. Leave no observation behind: Real-time correction for vla action chunks. *arXiv preprint arXiv:2509.23224*, 2025.

[17] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. Vlash: Real-time vlas via future-state-aware asynchronous inference. *arXiv preprint arXiv:2512.01031*, 2025.

[18] Figure AI. Helix: A vision-language-action model for generalist humanoid control. https://www.figure.ai/news/helix, 2025.

[19] Jianke Zhang, Yanjiang Guo, Xiaoyu Chen, Yen-Jen Wang, Yucheng Hu, Chengming Shi, and Jianyu Chen. Hirt: Enhancing robotic control with hierarchical robot transformers. *arXiv preprint arXiv:2410.05273*, 2024.

[20] Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, et al. Hume: Introducing system-2 thinking in visual-language-action model. *arXiv preprint arXiv:2505.21432*, 2025.

[21] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[22] Tony Z Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. Learning fine-grained bimanual manipulation with low-cost hardware. *arXiv preprint arXiv:2304.13705*, 2023.

[23] Jiayuan Gu, Sean Kirmani, Paul Wohlhart, Yao Lu, Montserrat Gonzalez Arenas, Kanishka Rao, Wenhao Yu, Chuyuan Fu, Keerthana Gopalakrishnan, Zhuo Xu, et al. Rt-trajectory: Robotic task generalization via hindsight trajectory sketches. *arXiv preprint arXiv:2311.01977*, 2023.

[24] Octo Model Team, Dibya Ghosh, Homer Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, et al. Octo: An open-source generalist robot policy. *arXiv preprint arXiv:2405.12213*, 2024.

[25] Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025.

[26] Dong Jing, Gang Wang, Jiaqi Liu, Weiliang Tang, Zelong Sun, Yunchao Yao, Zhenyu Wei, Yunhui Liu, Zhiwu Lu, and Mingyu Ding. Mixture of horizons in action chunking. *arXiv preprint arXiv:2511.19433*, 2025.

[27] Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. A survey on efficient vision-language-action models. *arXiv preprint arXiv:2510.24795*, 2025.

[28] Wenhao Sun, Sai Hou, Zixuan Wang, Bo Yu, Shaoshan Liu, Xu Yang, Shuai Liang, Yiming Gan, and Yinhe Han. Dadu-e: Rethinking the role of large language model in robotic computing pipelines. *Journal of Field Robotics*, 2026.

[29] Yunchao Ma, Yizhuang Zhou, Yunhuan Yang, Tiancai Wang, and Haoqiang Fan. Running vlas at real-time speed. *arXiv preprint arXiv:2510.26742*, 2025.

[30] Yiyang Huang, Yuhui Hao, Bo Yu, Feng Yan, Yuxin Yang, Feng Min, Yinhe Han, Lin Ma, Shaoshan Liu, Qiang Liu, et al. Dadu-corki: Algorithm-architecture co-design for embodied ai-powered robotic manipulation. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pages 327–343, 2025.

[31] Michael Davies, Neal Crago, Karthikeyan Sankaralingam, and Christos Kozyrakis. Liminal: Exploring the frontiers of llm decode performance. *arXiv preprint arXiv:2507.14397*, 2025.

[32] Abhimanyu Bambhaniya, Ritik Raj, Geonhwa Jeong, Souvik Kundu, Sudarshan Srinivasan, Suvinay Subramanian, Midhilesh Elavazhagan, Madhu Kumar, and Tushar Krishna. Demystifying ai platform design for distributed inference of next-generation llm models. *arXiv preprint arXiv:2406.01698*, 2024.

[33] Amey Agrawal, Nitin Kedia, Jayashree Mohan, Ashish Panwar, Nipun Kwatra, Bhargav S Gulavani, Ramachandran Ramjee, and Alexey Tumanov. Vidur: A large-scale simulation framework for llm inference. *Proceedings of Machine Learning and Systems*, 6:351–366, 2024.

[34] Jaehong Cho, Minsu Kim, Hyunmin Choi, Guseul Heo, and Jongse Park. Llmservingsim: A hw/sw co-simulation infrastructure for llm inference serving at scale. In *2024 IEEE International Symposium on Workload Characterization (IISWC)*, pages 15–29. IEEE, 2024.

[35] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, et al. Llm inference unveiled: Survey and roofline model insights. *arXiv preprint arXiv:2402.16363*, 2024.

[36] Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, and Vidushi Dadu. Rago: Systematic performance optimization for retrieval-augmented generation serving. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pages 974–989, 2025.

[37] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. Splitwise: Efficient generative llm inference using phase splitting. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 118–132. IEEE, 2024.

[38] Generalist AI Team. Gen-0: Embodied foundation models that scale with physical interaction. *Generalist AI Blog*, 2025. https://generalistai.com/blog/preview-uqlxvb-bb.html.

[39] Huiwon Jang, Sihyun Yu, Heeseung Kwon, Hojin Jeon, Younggyo Seo, and Jinwoo Shin. Contextvla: Vision-language-action model with amortized multi-frame context. *arXiv preprint arXiv:2510.04246*, 2025.

[40] Hao Shi, Bin Xie, Yingfei Liu, Lin Sun, Fengrong Liu, Tiancai Wang, Erjin Zhou, Haoqiang Fan, Xiangyu Zhang, and Gao Huang. Memoryvla: Perceptual-cognitive memory in vision-language-action models for robotic manipulation. *arXiv preprint arXiv:2508.19236*, 2025.

[41] Zixuan Wang, Bo Yu, Junzhe Zhao, Wenhao Sun, Sai Hou, Shuai Liang, Xing Hu, Yinhe Han, and Yiming Gan. Karma: Augmenting embodied ai agents with long-and-short term memory systems. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8. IEEE, 2025.

[42] Ruoshi Wen, Jiajun Zhang, Guangzeng Chen, Zhongren Cui, Min Du, Yang Gou, Zhigang Han, Junkai Hu, Liqun Huang, Hao Niu, et al. Dexterous teleoperation of 20-dof bytedexter hand via human motion retargeting. *arXiv preprint arXiv:2507.03227*, 2025.

[43] Clemens C Christoph, Maximilian Eberlein, Filippos Katsimalis, Arturo Roberti, Aristotelis Sympetheros, Michel R Vogt, Davide Liconti, Chenyu Yang, Barnabas Gavin Cangan, Ronan J Hinchet, et al. Orca: An open-source, reliable, cost-effective, anthropomorphic robotic hand for uninterrupted dexterous task learning. *arXiv preprint arXiv:2504.04259*, 2025.

[44] Ricky KP Mok, Hongyu Zou, Rui Yang, Tom Koch, Ethan Katz-Bassett, and Kimberly C Claffy. Measuring the network performance of google cloud platform. In *Proceedings of the 21st ACM internet measurement conference*, pages 54–61, 2021.

[45] Igor Sfiligoi, John Graham, and Frank Wuerthwein. Characterizing network paths in and out of the clouds. In *EPJ Web of Conferences*, volume 245, page 07059. EDP Sciences, 2020.

A Detailed System and Model Parameters

In this section, we show the detailed hardware performance configuration used in our evaluation and the model parameters of π_0 in Table 11.

Table 10: Hardware specifications for the GPUs used in our evaluation.

Hardware	**FP32**	**BF16/FP16**	**INT8**	**Memory**	**Memory BW**
Jetson Thor	100 TFLOP/s	400 TFLOP/s	800 TOP/s	128 GB	270 GB/s
RTX 4090	83 TFLOP/s	165 TFLOP/s	330 TOP/s	24 GB	1008 GB/s
A100	20 TFLOP/s	312 TFLOP/s	624 TOP/s	80 GB	2039 GB/s
H100	67 TFLOP/s	989 TFLOP/s	1979 TOP/s	80 GB	3350 GB/s
B100	60 TFLOP/s	1750 TFLOP/s	3500 TOP/s	192 GB	8000 GB/s

Table 11: Parameter specifications for π_0 model components (without vocabulary table).

Component	**Layers**	**Hidden Dim**	**Interm. Dim**	**Q Heads**	**KV Heads**	**Params**
Vision Encoder	27	1,152	4,304	16	16	411.19M
VLM Backbone	18	2,048	16,384	8	1	1.98B
Action Expert	18	1,024	4,096	8	1	292.63M