Beyond LLM Serving: Characterizing Vision-Language-Action Workloads for Embodied AI System Design

Seonghun Jung
KAIST
Daejeon, Republic of Korea
shjung@casys.kaist.ac.kr

Sieun Moon
KAIST
Daejeon, Republic of Korea
moon2121@kaist.ac.kr

Jiyoung Jeong
KAIST
Daejeon, Republic of Korea
ji_0.jeong@kaist.ac.kr

Jimin Lee
KAIST
Daejeon, Republic of Korea
jimin-lee@kaist.ac.kr

Jaehyuk Huh
KAIST
Daejeon, Republic of Korea
jhhuh@kaist.ac.kr

Abstract

Vision-language-action (VLA) models translate multimodal observations into low-level robot actions. During robot operation, each control period sets an inference deadline, and overruns leave the robot acting on stale observations, reducing task success. Meeting this deadline motivates on-device or nearby edge execution, where a single robot requires batch-1 inference outside the design point of LLM serving systems. Although VLA architectures combine familiar vision-language, autoregressive, and diffusion-style components, their runtime behavior in this batch-1 control setting remains uncharacterized. We characterize four representative VLA models on an edge GPU server and two onboard SoCs, using single-inference profiling and 43,200 closed-loop episodes. Action tensor dimensionality determines whether a stage is memory- or compute-bound, platform balance can shift that bottleneck, and GPU frequency scaling yields a platform-dependent energy–latency sweet spot. In closed-loop operation, overlapping inference with action execution creates an accuracy–speed–energy tradeoff, and no configuration is Pareto-dominant across deployment SLOs. These results guide joint design of VLA model architectures, hardware, and runtime policies.

CCS Concepts: • **Computer systems organization → Robotics**; *Special purpose systems*.

Keywords: vision-language-action models, embodied manipulation, closed-loop inference, workload characterization



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

ASPLOS '27, Heraklion, Greece
© 2027 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-3012-2/2027/04
https://doi.org/10.1145/3845814.3850089

ACM Reference Format:
Seonghun Jung, Sieun Moon, Jiyoung Jeong, Jimin Lee, and Jaehyuk Huh. 2027. Beyond LLM Serving: Characterizing Vision-Language-Action Workloads for Embodied AI System Design. In *Proceedings of the 32nd ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1 (ASPLOS '27), April 11–15, 2027, Heraklion, Greece*. ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3845814.3850089

1 Introduction

Embodied action generation models, often referred to as vision-language-action (VLA) models [5, 7, 8, 11, 21, 22, 25, 27, 33, 44, 45, 52, 60], translate multimodal observations into low-level robot control signals for manipulation tasks. Real-world manipulation requires smooth and continuous motion, so models designed for real robots increasingly produce not a single next command but an *action chunk*, a fixed-length sequence of future commands that together form a short trajectory segment [19, 59]. To generate such chunks, these models typically combine a large perception backbone with a specialized *action-generation head*, a neural network that converts the backbone's latent representation into the action chunk. Recent architectures generate action chunks through iterative denoising over compact action tensors [5, 7, 8, 11, 25, 33, 44, 52], single-pass parallel decoding [21, 27, 45], or unified diffusion over spatiotemporal video latents [22]. These action-generation choices produce different latency and resource profiles that shape deployment decisions.

During deployment, these models operate in a closed loop: the robot continuously senses the environment, invokes the model, executes the resulting commands, and updates its next input based on the physical outcome [31, 42]. The control period, roughly 33 ms for contact-rich manipulation, therefore acts as a recurring deadline rather than merely a per-request latency target: inference that falls behind it leaves the robot acting on stale observations, and task success degrades. Meeting this deadline favors on-device hardware

or nearby consumer-grade edge servers because remote serving adds round-trip delay to the same latency budget. In the single-robot setting studied here, inference therefore runs at batch size one.

LLM serving research has produced a mature toolbox, including continuous batching, paged KV caches, and prefill-decode disaggregation, designed around concurrent autoregressive requests and token-level execution [24, 62]. These assumptions hold only partially in this setting: batch-1 serving provides no request concurrency, the action-chunk models we evaluate produce their chunks non-autoregressively, and the dominant stage varies with the architecture rather than with an autoregressive phase. Standard LLM serving optimizations therefore apply selectively (Section 6). VLA model families draw on familiar vision-language, autoregressive, and diffusion-style components, but how these components interact under batch-1 closed-loop serving has not been characterized.

To fill this gap, we conduct a system-level workload characterization of VLA serving. We study four representative models, namely GR00T N1.6 [33], $\pi_{0.5}$ [4], OpenVLA-OFT [21], and Cosmos-Policy [22], that span three action-generation paradigms: iterative denoising, parallel decoding, and video generation. We evaluate these models on three platforms that span over an order of magnitude in power budget: RTX 4090, Jetson Thor, and Jetson AGX Orin [37]. Our methodology combines four measurement techniques that target different levels of the system stack: stage-level latency decomposition, roofline-based bottleneck attribution, DVFS [41, 47, 51] sensitivity analysis, and closed-loop deployment on MuJoCo [48] with LIBERO [30] benchmarks, totaling 43,200 closed-loop episodes.

Our single-inference analysis shows that action tensor dimensions separate memory-bound stages from compute-bound ones. Memory-bound stages tolerate lower GPU clocks. Reducing memory bandwidth, however, raises latency and energy on both SoCs. The same bottleneck differences shape throughput per joule in deployment. Overlapping inference with action execution [6, 44, 46] creates an accuracy–speed–energy tradeoff. Accuracy degradation depends on model architecture, relative speedup is stable across platforms, and energy scaling depends on platform characteristics.

To our knowledge, this is the first system-level workload characterization to connect stage-level latency and roofline analysis with cross-platform DVFS sensitivity and closed-loop deployment. Our contributions are as follows:

Cross-model characterization. We show that VLA serving has no uniform bottleneck profile: `torch.compile` speedups differ by over 5× between action-head-dominated and backbone-dominated models. This implies that compiler and kernel-level optimizations must be applied selectively per architecture rather than uniformly.

Microarchitectural analysis. We show that the memory- versus compute-bound split is determined by action tensor dimensionality. Denoising-head GEMMs fall below the 163.9 FLOP/B ridge point, whereas large-sequence backbone and video-generation DiT GEMMs exceed it. This provides a basis for accelerator bandwidth-to-compute provisioning in VLA-targeted hardware.

Platform and energy insights. We show that GPU frequency scaling yields an energy sweet spot whose benefit depends on platform balance, as the same sweep saves 24–32% energy on Orin but only 1–4% on Thor, while memory-bandwidth throttling provides no energy benefit on either SoC. We further show that, when evaluated in isolation, assigning each inference phase its own GPU frequency can reduce both latency and energy when the phases exhibit different frequency sensitivities. However, in end-to-end execution, the overhead of switching GPU frequencies between phases can offset these gains.

Closed-loop tradeoff analysis. We quantify a three-way tradeoff in which accuracy degrades by up to 56 percentage points, speed improves by up to 3.9×, and energy scales by up to 5.1× under increasing overlap, with no Pareto-dominant configuration across SLOs. This establishes that runtime systems must support SLO-aware operating-point selection rather than static configurations.

These observations motivate bottleneck-aware co-design across model architecture, hardware platform, and runtime policy for embodied AI system design.

2 Background & Motivation

2.1 Embodied Action Generation Models

Embodied action generation models produce low-level robot actions from visual observations and language instructions [31, 42]. Recent architectures first encode the multimodal observation and the instruction into an internal representation, and then generate actions from it. They differ in the architecture that builds this representation and in the action-generation interface.

In the VLM-based models, the *perception backbone* typically combines a vision encoder (e.g., ViT [13], SigLIP [58]) with a language model (e.g., Llama 2 [49], PaliGemma [2]) to produce contextual embeddings that capture both spatial and semantic information. This stage performs a single forward pass through billions of parameters, inheriting the computational profile of large vision-language models.

The action-generation interface takes one of two forms: a sequence of discretized *action tokens* generated autoregressively, or an *action chunk*, a fixed-length sequence of future robot commands produced jointly in one inference [19, 59].

Autoregressive action-token interface. Token-based VLAs such as RT-2 [64] and vanilla OpenVLA [23] represent a robot action as a sequence of discretized action tokens generated sequentially by the backbone's language-model

Table 1. Latency decomposition and NCU roofline classification of autoregressive OpenVLA on RTX 4090 in eager mode.

Stage	Latency (ms)	NCU GPU time (%)	
		Memory-bound	Compute-bound
Vision encoder	16.03	23.9	76.1
LLM prefill	43.50	13.6	86.4
Cached decode (6 passes)	140.90	100.0	0.0
Other + preprocessing	9.13	–	–
E2E	209.56	–	–

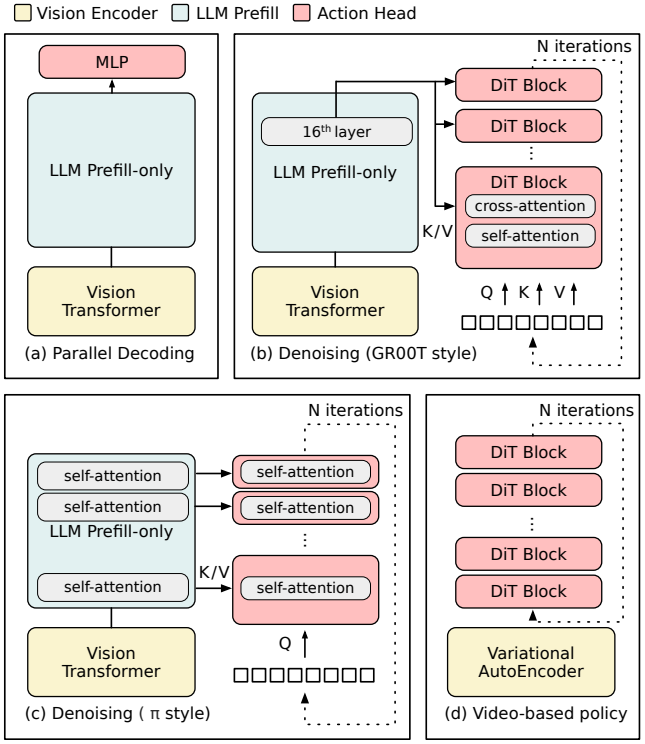
decoder. In the OpenVLA formulation we profile, the target robot's action space has D dimensions, and each dimension maps to one token. One action therefore requires D sequential token-generation steps: one prefill step followed by $D-1$ cached-decode passes.

This sequential decode path is difficult to amortize under high-rate control. On the RTX 4090, its six cached-decode passes take 140.90 ms in our 7-dimensional setting. This is 3.2× the LLM prefill latency and accounts for 67% of the 209.56 ms inference (Table 1). Cached decode therefore dominates the latency of autoregressive action generation.

The same decode path exhibits the familiar LLM resource split. Prefill is mostly compute-bound, whereas cached decode is entirely memory-bound (Section 4.2). However, the fixed decode horizon leaves little opportunity for KV-cache reuse. The cache grows by only $D-1$ positions within an action and is discarded when the action completes.

Because this interface emits one D-token action per inference, producing K future actions repeats those D sequential steps K times. OpenVLA-OFT [21] instead emits an entire K-action chunk in one pass. Its reported action throughput on LIBERO is roughly 26× that of autoregressive OpenVLA. This contrast motivates the action-chunk interface, which amortizes one inference over K future commands. Action-chunk models use this interface but generate each chunk through one of three representative paradigms.

Iterative denoising heads. This paradigm generates the action chunk through a multi-step denoising process that invokes the same neural network N times (typically 4–10) per action prediction. Representative models refine a random initial tensor into the action trajectory, using flow-matching objectives [29] (GR00T [33], π_0 [5], SmolVLA [44], GR-3 [8], GraspVLA [11]) or denoising diffusion [10, 18] (RoboDual [7], CogACT [25], DiffusionVLA [52]). Regardless of the training objective, the computational signature is shared: the state being denoised is a compact action trajectory tensor of shape [B, action_horizon, action_dim] (batch size 1 in single-robot deployment), so each denoising step runs the network over this short, low-dimensional sequence.



† DiT: Diffusion Transformer

Figure 1. Architectures and profiling decomposition of the four action-chunk models evaluated in closed loop.

Parallel decoding heads. This paradigm generates the entire action chunk in a single forward pass using parallel token prediction or regression. Representative models include OpenVLA-OFT [21] (L1 regression MLP), PD-VLA [45], and CogVLA [27].

Unified video generation. This paradigm embeds robot actions in the latent sequence of a pretrained video diffusion model, treating action generation as a joint video–action prediction problem within a shared latent space [16, 26]. Unlike the VLM-based models above, Cosmos-Policy [22] attaches no separate action head to a perception backbone: it adapts Cosmos-Predict2 [32] into a unified policy in which one generative model predicts actions together with future visual observations.

Cosmos-Policy reuses the spatiotemporal VAE tokenizer of its pretrained video backbone to encode image sequences into compact latent frames. It then injects robot proprioception, action chunks, and future observations into dedicated slots of this latent sequence. A DiT (Diffusion Transformer, a Transformer operating on latent image patches) [40] denoises the target slots conditioned on the current-observation frames [22]. Actions are then read directly from the denoised action slots rather than decoded from generated RGB frames. The resulting tensors are far larger than the compact action trajectories of the action-only denoising heads

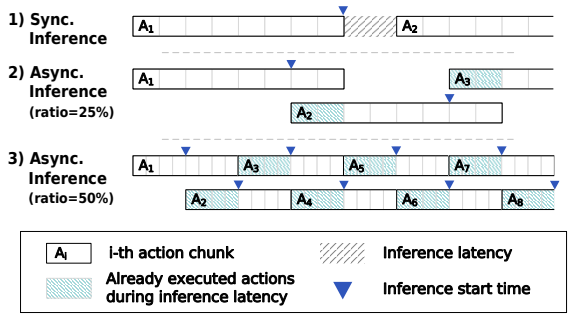


Figure 2. Synchronous and asynchronous inference. In synchronous mode, the robot idles during inference (top). Asynchronous overlap initiates the next inference before the current chunk completes, with the overlap percentage governing when the next request is triggered (middle and bottom).

above: 9 temporal frames at 28×28 spatial resolution yield 9×28×28 = 7,056 token positions of 16 channels each.

For cross-model profiling, we separate VAE encoding from DiT denoising and group the latter under the action-head stage, because the action-generation computation occurs within the DiT. This is an operational decomposition for stage-level comparison: the DiT remains part of Cosmos-Policy's unified video-diffusion architecture and also processes the other latent slots.

Figure 1 illustrates the four action-chunk models evaluated in closed loop: GR00T N1.6 (flow-matching DiT), $\pi_{0.5}$ [4] (flow-matching Gemma Expert), OpenVLA-OFT (L1 regression MLP), and Cosmos-Policy (video-generation DiT). These four models span the three action-generation paradigms.

2.2 Closed-Loop Policy Serving

When these policies drive a robot in closed loop, each inference must complete within a fixed control period, typically 33 ms or less for the 30 Hz rates required by contact-rich manipulation tasks. Current policy models are far from meeting this budget [21, 50, 54, 61]: in our measurements, inference takes 117–396 ms on an edge-server GPU (RTX 4090) and 304–2603 ms on an edge SoC (Jetson AGX Orin), yielding effective control frequencies of only 2.5–8.5 Hz and 0.4–3.3 Hz, respectively.

In a synchronous execution scheme, the control loop captures a sensor observation, invokes the policy model, and dispatches the resulting action chunk to the actuator before the next cycle begins [4, 5, 14, 21, 33, 64]. This design maximizes observation freshness but, when inference overruns the control period, forces the pipeline to stall, repeating stale actions and introducing visible discontinuities in the robot's motion trajectory. Unlike offline evaluation, where a simulator simply pauses during inference, real-world deployment offers no mechanism to stop time, so every millisecond of excess latency directly degrades control quality [6].

To bridge this gap, deployed systems employ *asynchronous overlap* [6, 44, 46]: the system issues the next inference request while the robot is still executing actions from the current chunk, hiding inference latency behind action execution (Figure 2).

However, asynchronous overlap introduces a tradeoff: because inference is initiated before the current chunk completes, the observation used for the next prediction ages while the robot continues executing [50]. We refer to this lag as *observation staleness*, the interval between observation capture and the execution of actions derived from it:

$$s = t_{\text{execute}} - t_{\text{capture}}. \tag{1}$$

More aggressive overlap enables faster control but forces the robot to act on increasingly stale observations, potentially degrading task accuracy [43]. We refer to this as the *accuracy–staleness tradeoff*.

2.3 Motivation

The workload described in Section 2.1 and Section 2.2 diverges from the assumptions that underlie LLM serving infrastructure in three ways. Each divergence motivates a distinct dimension of system-level characterization.

M1. Action-generation heterogeneity requires stage-level bottleneck analysis. LLM serving analyses decompose cost along the autoregressive phases of prefill and decode. That decomposition carries over to the action-token interface, which uses the same prefill–decode mechanism. The action-chunk models described in Section 2.1, however, use heterogeneous iterative-denoising, parallel-decoding, and video-generation stages, which vary further within each paradigm through tensor shape and GEMM dimension choices. Characterizing them requires a stage-level decomposition organized around action-generation design.

M2. Latency and task accuracy are coupled, not orthogonal. In LLM serving, latency is a service-level property and response accuracy is a model property, and the two can be optimized independently. The accuracy–staleness tradeoff described in Section 2.2 removes this separation: per-inference latency affects the feasible overlap, which changes observation staleness and task success. Understanding this coupling requires measurement at the deployment level, not the per-inference level alone.

M3. Deployment frequently spans heterogeneous platforms. LLM serving has primarily targeted datacenter-class GPUs. Embodied action generation often runs on edge SoCs in addition to server GPUs, because autonomous operation may occur where network connectivity is unreliable or round-trip latency exceeds the control period. These platforms differ in compute, bandwidth, and power, so a characterization on a single tier does not readily transfer to others.

Table 2. Architecture overview of evaluated models.

Model	Backbone	Action Head	Config[1]
GR00T N1.6	Cosmos-Reason VLM (Eagle ViT + LLM)	DiT (Flow matching)	4/3B/16
$\pi_{0.5}$	PaliGemma VLM (SigLIP+Gemma 2B)	Gemma 300M (Flow matching)	10/3B/10
OpenVLA-OFT	Prismatic VLM (SigLIP+DINOv2+ Llama 2)	MLP (L1 regression)	N/A/7B/8
Cosmos-Policy	Wan2.1 VAE	DiT (Diffusion)[2]	5/2B/16

[1]Config = Steps / ~Params / Chunk. *Steps*: denoising iterations; *Chunk*: action steps per inference.
[2]Cosmos-Policy employs a unified video diffusion architecture (Cosmos-Predict2-2B); VAE and DiT are listed separately for profiling clarity.

Table 3. Hardware platforms and software stacks used in evaluation.

Platform	Tier	Memory	Power	SW Stack
Jetson AGX Orin	Power-efficient SoC	64 GB LPDDR5	15–60 W	JetPack 6.2
Jetson Thor	High-performance SoC	128 GB LPDDR5X	70–130 W	JetPack 7.0
RTX 4090	Edge server	24 GB GDDR6X	450 W	CUDA 13.1

Table 4. Measurement methodology summary. Each analysis targets a different level of the system stack; detailed configurations are deferred to the corresponding sections.

Analysis	Metric	Tool	Sec.
Latency decomp.	Wall-clock per stage (ms)	`perf_counter` + `cuda.sync`	4.1
Roofline profiling	GPU kernel time, FLOPs, DRAM bytes	NVIDIA NCU (v2025.2.0)	4.2
DVFS sensitivity	E2E latency per freq. point	`perf_counter` + `sysfs`	4.3
Closed-loop	Success rate, measured energy	MuJoCo + NVML / `sysfs`	5

3 Methodology

3.1 Models and Benchmarks

Table 2 lists the four embodied action generation models we evaluate. The set is chosen along two axes. First, it spans both action-generation mechanisms: iterative denoising (GR00T N1.6 and $\pi_{0.5}$ over action tensors, Cosmos-Policy over video latents) and single-pass parallel decoding (OpenVLA-OFT), which impose different compute and control-loop profiles. Second, it spans backbone families from standard vision-language models [2, 28, 49] to video-generation models [32], exposing how pretraining objective shapes deployment behavior. The goal is to cover diverse compute, memory, and control-loop behaviors rather than to maximize benchmark breadth.

For the four models we evaluate end to end, we use the LIBERO-finetuned checkpoints from each model's official repository rather than training our own variants. Our measurements therefore reflect the deployment behavior of community-adopted configurations. We additionally profile the public `openvla-7b` checkpoint as an autoregressive action-token baseline and exclude it from the closed-loop evaluation for its limited action throughput (Section 2.1).

We evaluate on four LIBERO [30] suites (40 tasks in total), which we organize into two task groups by task horizon for our analysis. We refer to LIBERO-10 as the *Long* group (10 long-horizon manipulation tasks with multi-step dynamic interactions), and we pool LIBERO-Goal, LIBERO-Spatial, and LIBERO-Object into the *Short* group (30 short-horizon, shorter-episode tasks). We report the groups separately to compare degradation across task horizons. Tasks are the sampling units, and success rate (SR) is the mean of task-level success rates. For latency, energy, and throughput, we first average successful episodes per task and then average tasks.

3.2 Platforms and Software

Table 3 covers an edge server and two onboard SoCs, with power budgets ranging from 15 W to 450 W. We designate the RTX 4090 as the primary platform for per-kernel roofline analysis [12, 17] because its discrete architecture exposes well-defined memory-bandwidth and compute-throughput ceilings, whereas the Jetson SoCs share DRAM between CPU and GPU and expose only aggregate bandwidth, complicating per-kernel attribution. Jetson Thor and AGX Orin [37] serve as deployment targets representing next- and current-generation mobile robots. All platforms run PyTorch 2.10. The RTX 4090 uses CUDA 13.1, while Thor and AGX Orin use JetPack 7.0 and 6.2 respectively, each bundling matched CUDA [34], cuDNN [9], and TensorRT [38] versions for its SoC generation.

3.3 Measurement Methodology

Table 4 summarizes measurement methodology by analysis, and Table 5 reports per-model compilation settings. All timing measurements insert GPU synchronization barriers (`cuda.synchronize`) at stage boundaries to attribute GPU work to the correct pipeline stage. Energy measurements cover the GPU on RTX 4090. On each SoC, they cover the compute-domain rails used by NVIDIA for module power capping: `VDD_GPU_SOC` and `VDD_CPU_CV` on Orin [35], and

Table 5. Compilation configuration used for DVFS analysis and closed-loop evaluation. Compilation mode refers to `torch.compile` mode; "eager" denotes no compilation. Stage-level decomposition and roofline profiling use eager mode throughout.

Model	Backbone	Action Head
GR00T N1.6	Default	Max-autotune
$\pi_{0.5}$	Max-autotune	Max-autotune
OpenVLA-OFT	Max-autotune*	N/A
Cosmos-Policy	Eager	Eager

*Includes prewarm for variable-length task prompts.

VDD_GPU and VDD_CPU_SOC_MSS on Thor [36]. These boundaries exclude total module input power. We therefore compare energy directly within each platform and use cross-platform values only for relative trends.

For reproducibility, all models use each repository's official LIBERO-finetuned checkpoint at its native input resolution, inference precision, and chunk length. The per-phase frequency assignment in Section 4.3.3 assigns clocks only to model-execution phases. Idle time is excluded because it is set by the control schedule rather than by a single inference, and it is evaluated as part of the closed-loop timing analysis in Section 5.

The per-model compilation settings in Table 5 reflect model-specific constraints that we document here for reproducibility. Cosmos-Policy runs in eager mode because compiling its TransformerEngine-based DiT slightly increases latency (Section 4.1.2) and its VAE tokenizer is not exposed as a compilable module. GR00T's backbone triggers tensor aliasing assertions under CUDA graph capture, so we apply `max-autotune` only to its action head and leave the backbone in default mode. OpenVLA-OFT compiles under `max-autotune`, but its task-description-dependent input length requires pre-warming all LIBERO task prompts before evaluation to cache each compilation variant. $\pi_{0.5}$ compiles cleanly under `max-autotune` across the full model.

These compiled configurations are used in analyses targeting deployment-realistic latency: DVFS sensitivity (Section 4.3) and closed-loop evaluation (Section 5). Stage-level latency decomposition (Section 4.1) and roofline profiling (Section 4.2) instead use eager mode, because compiler-induced kernel fusion and CUDA graph capture alter operator boundaries and obscure the per-kernel bottleneck structure those analyses target. Section 4.1.2 explicitly compares both modes.

4 Single-Inference Characterization

We characterize four action-chunk models through stage-level latency decomposition (§4.1), roofline bottlenecks (§4.2), and edge-SoC DVFS sensitivity (§4.3). All measurements use random inputs matching each model's expected tensor shapes.

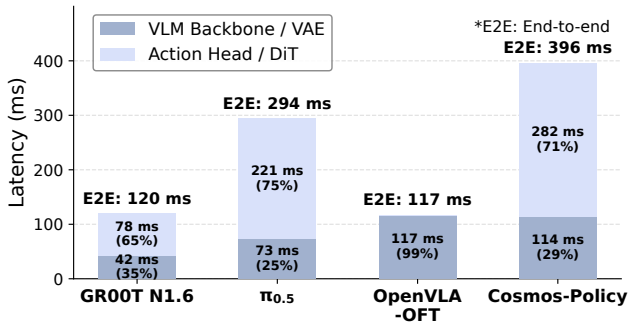


Figure 3. End-to-end (E2E) latency breakdown in eager mode.

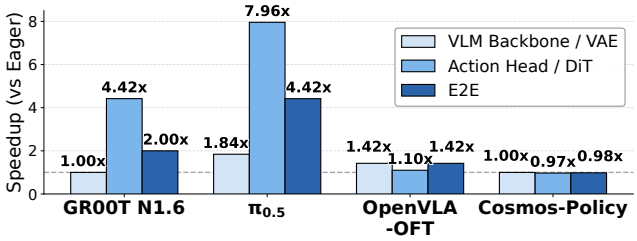


Figure 4. Per-stage compilation speedup over eager execution by model.

4.1 Stage-Level Latency Decomposition of Action-Chunk Models

4.1.1 Eager-mode breakdown. Figure 3 decomposes the end-to-end inference latency of the four action-chunk models into backbone and action-head contributions on the RTX 4090. Action generation accounts for 65% of GR00T N1.6's latency, 75% for $\pi_{0.5}$, and 71% for Cosmos-Policy. OpenVLA-OFT is the exception: its single-pass MLP head completes in 0.6 ms, making the 7 B Llama 2 backbone the sole bottleneck (99%).

4.1.2 Per-stage compilation. Figure 4 compares eager execution with independent compilation of each eligible stage. Compiled stages use `max-autotune`, except GR00T N1.6's backbone, which uses the default mode because of its CUDA-graph constraint (Section 3.3). Cosmos-Policy's VAE tokenizer is not exposed as a compilable module, so it remains eager. Compilation accelerates the diffusion-based action heads most: 7.96× for $\pi_{0.5}$ and 4.42× for GR00T N1.6. OpenVLA-OFT instead gains 1.42× from backbone compilation. For Cosmos-Policy, compiling the TransformerEngine-based DiT slightly increases latency (0.97×). The deployment configurations in Table 5 therefore retain compilation only for stages where it improves latency.

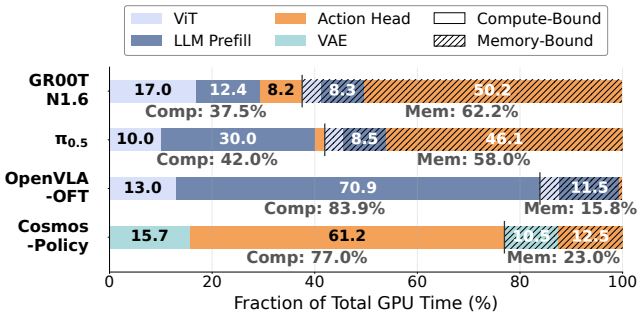


Figure 5. Roofline classification by stage on RTX 4090. Mem = memory-bound fraction of GPU time; Comp = compute-bound fraction.

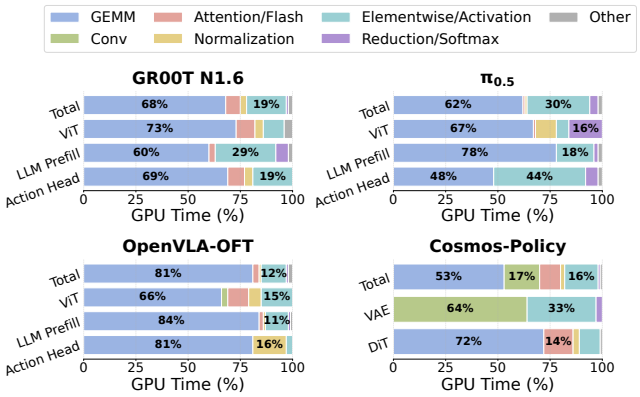


Figure 6. Operator breakdown by stage and model on the RTX 4090.

Takeaway 1. *VLA pipeline architecture determines both the dominant latency stage and where compilation helps. The iterative action head dominates and benefits most from compilation in GR00T N1.6 and $\pi_{0.5}$; in OpenVLA-OFT, the backbone does. Compilation targets should therefore be selected per architecture.*

4.2 Compute–Memory Bottleneck Analysis of Action-Chunk Models

Roofline analysis [12, 17, 53] classifies each GPU kernel as memory- or compute-bound from its arithmetic intensity (AI = FLOPs/Bytes). The ridge point is the ratio of peak compute throughput to peak memory bandwidth ($\pi_{\text{peak}}/\beta_{\text{peak}}$). Kernels whose arithmetic intensity falls below the ridge point are memory-bound because memory bandwidth limits performance before the compute units saturate. The RTX 4090 provides 165.2 TFLOP/s of FP16 throughput and 1.008 TB/s of memory bandwidth, giving a ridge point of 163.9 FLOP/B.

Table 1 applies the same classification to the autoregressive baseline. Figure 5 classifies GPU time by bottleneck type for each pipeline stage of the four action-chunk models. GR00T N1.6 and $\pi_{0.5}$ are predominantly memory-bound (62% and 58% of GPU time), with the memory-bound fraction concentrated in their action heads. OpenVLA-OFT and Cosmos-Policy are predominantly compute-bound (84% and 77%), dominated by the Llama 2 backbone and video-generation DiT, respectively.

GEMMs dominate GPU time in every model and stage (Figure 6), yet stages differ in the resource that limits them. Individual GEMM size therefore determines whether a stage is memory- or compute-bound.

Figure 7 plots per-GEMM arithmetic intensity against operation size. GR00T N1.6 and $\pi_{0.5}$ action-head GEMMs concentrate below the ridge point, while OpenVLA-OFT backbone and Cosmos-Policy DiT GEMMs cluster above it. The intensity gap is determined by the tensor dimensions processed by each stage. GR00T N1.6's DiT processes a short sequence of action and state tokens. At batch size one, its linear layers reuse their weights across only a few tokens, limiting arithmetic intensity. The denoising loop repeats these low-intensity operations at each step. $\pi_{0.5}$ similarly processes a short suffix sequence at each step, and its action-head GEMMs also fall below the ridge point. OpenVLA-OFT processes all tokens in full self-attention through a 7 B backbone, and Cosmos-Policy denoises a spatiotemporal latent with 7,056 effective positions. Both produce GEMMs above the ridge point.

Takeaway 2. *The memory/compute-bound distinction arises from tensor dimensionality: iterative denoising over compact action tensors is bottlenecked by memory bandwidth, while large-sequence processing saturates compute units.*

4.3 Cross-Platform Resource Sensitivity on Edge SoCs

The roofline analysis above was conducted on a discrete GPU (RTX 4090) with separate compute and memory subsystems. Edge SoCs share a unified memory hierarchy and expose three independently clocked domains, each scalable via DVFS [41, 47, 51]: CPU cores, GPU, and the EMC (external memory controller, which governs DRAM bandwidth) [35, 36]. This subsection examines whether the roofline-predicted bound types manifest as measurable resource sensitivity on Jetson Thor and AGX Orin.

Definitions. We define the *sensitivity* of a pipeline stage (or end-to-end inference) to a clock domain as the normalized latency when that domain is set to its minimum frequency, with all other domains held at maximum:

$$S_{\text{domain}} = \frac{T_{\text{domain}=f_{\min}}}{T_{\text{all}=f_{\max}}}. \tag{2}$$

A higher value indicates greater dependence on that resource. To compare the two dominant hardware resources within a single scalar, we define the *GPU/EMC sensitivity ratio*:

$$R = \frac{S_{\text{GPU}}}{S_{\text{EMC}}}. \tag{3}$$

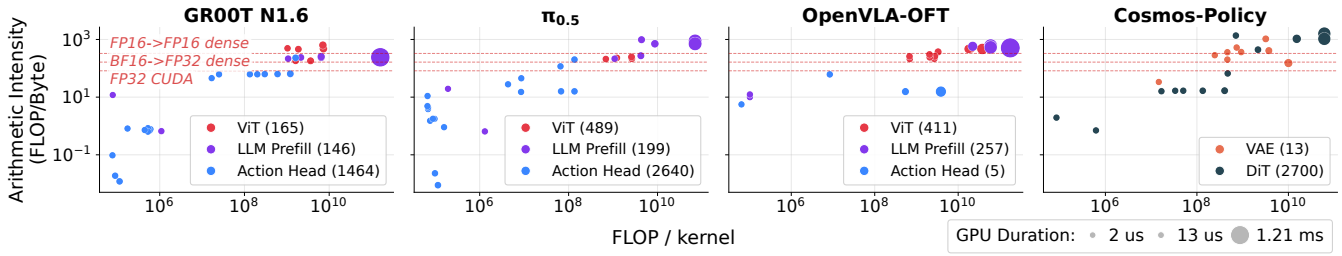


Figure 7. GEMM size and arithmetic intensity on the RTX 4090 roofline.

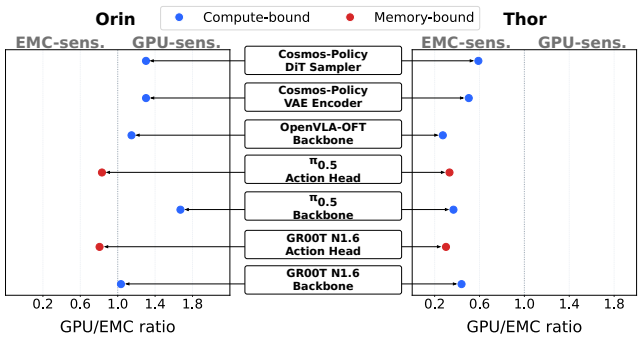


Figure 8. Per-stage GPU/EMC sensitivity ratio R (Eq. 3) on Orin and Thor. Points are colored by the roofline-predicted bottleneck (memory- or compute-bound) from RTX 4090 eager-mode profiling.

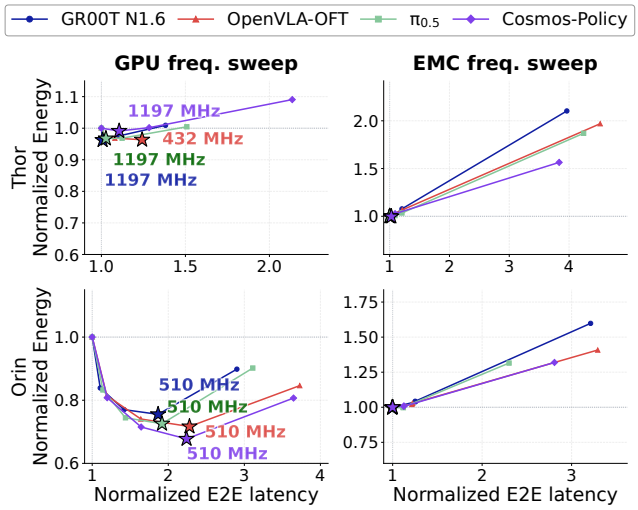


Figure 9. Energy–latency tradeoffs under GPU and EMC frequency sweeps on Jetson AGX Orin and Jetson Thor.

A ratio $R < 1$ indicates that the stage is more sensitive to memory bandwidth (EMC) than to compute throughput (GPU), while $R > 1$ indicates GPU-compute dominance.

Experimental setup. All measurements use compiled execution with the per-model configurations listed in Table 5. GPU and EMC frequencies are swept independently while all other domains remain at maximum. The swept ranges are: Orin GPU 1300 → 306 MHz (4.2×), Orin EMC 3199 → 665.6 MHz (4.8×), Thor GPU 1575 → 432 MHz (3.6×), and Thor EMC 4266 → 665.6 MHz (6.4×). Because the two platforms differ in frequency range, absolute sensitivity values are not directly comparable across platforms, so we use the ratio R to compare resource binding within and across platforms.

4.3.1 Per-Stage Sensitivity Ratio. Figure 8 plots the GPU/EMC sensitivity ratio R for each pipeline stage on both platforms, with points colored by the roofline-predicted bottleneck (memory- or compute-bound) obtained from RTX 4090 eager-mode profiling (Section 4.2).

On Orin, the sensitivity ratio separates memory-bound from compute-bound stages around $R \approx 1$. Memory-bound action heads (GR00T N1.6 R=0.81, $\pi_{0.5}$ R=0.83) exhibit stronger relative EMC sensitivity than all compute-bound stages (backbones, Cosmos-Policy VAE and DiT), which range from R=1.03 to R=1.67. The roofline-predicted ordering is also observed in DVFS-derived resource sensitivity on this platform.

On Thor, absolute R values drop below 1 for all stages, reflecting the platform's higher compute-to-bandwidth ratio: GPU throttling from 1575 to 432 MHz yields only 1.3–2.3× slowdown across a 3.6× frequency range, while EMC throttling to 665.6 MHz causes 3.3–4.7× slowdown. Despite this platform-wide shift, the relative ordering among stages largely tracks the roofline classification: stages with sub-ridge arithmetic intensity (GR00T N1.6 action head R=0.30, $\pi_{0.5}$ action head R=0.33) show lower R than most compute-bound stages (GR00T N1.6 backbone R=0.44, Cosmos-Policy VAE R=0.50, DiT R=0.59). The ordering is not perfect. The OpenVLA-OFT backbone (R=0.27) falls below both memory-bound action heads. The overall trend, however, indicates that per-kernel arithmetic intensity continues to modulate relative EMC sensitivity even when the platform places all stages in an EMC-dominant regime.

Together, the Orin and Thor results show that platform balance shifts absolute sensitivity, while arithmetic intensity still organizes the broad stage-level trend.

Table 6. Measured GPU-frequency transition latency and its share of baseline E2E latency for two transitions per inference. OpenVLA-OFT has no intra-inference transition and is omitted.

Platform	Transition latency		Share of E2E latency		
	Per transition	Per inference	GR00T N1.6	$\pi_{0.5}$	Cosmos-Policy
Orin	3.5 ms	7.0 ms	2.5%	1.2%	0.3%
Thor	8.6 ms	18.0 ms	13.6%	7.8%	1.7%

Takeaway 3. *Whether a stage is limited by memory bandwidth or by compute throughput depends on both its arithmetic intensity and the platform's compute-to-bandwidth ratio. The same stage can therefore hit different bottlenecks across platforms, while the relative ordering of stages by intensity is largely preserved.*

4.3.2 Energy–Latency Tradeoff under Frequency Scaling.

Per-inference energy equals average power multiplied by execution time. Figure 9 plots normalized per-inference energy against normalized end-to-end latency for GPU and EMC frequency sweeps on both platforms.

GPU throttling traces a convex energy–latency curve. On Orin, all four models reach minimum energy at 510 MHz, saving 24–32% of per-inference energy at 1.9–2.2× latency cost. The minimum exists because power drops faster than latency rises while memory-bound stages are less sensitive to GPU clock reduction. Below 510 MHz, latency growth in compute-bound stages overtakes the power saving. On Thor, the curve is much flatter: the energy-optimal point (1197 MHz for three models, 432 MHz for OpenVLA-OFT) yields only 1–4% savings, consistent with the observation that GPU throttling has limited impact on latency across all stages on this platform.

EMC throttling offers no energy benefit. Energy rises monotonically on both platforms as EMC frequency decreases, reaching +32% to +60% on Orin and +56% to +110% on Thor at minimum frequency. Bandwidth starvation inflates latency across all stages, and this increase outweighs any power reduction from lower DRAM clock.

Takeaway 4. *Among the evaluated DVFS controls, only GPU frequency scaling yields an energy–latency sweet spot. The attainable energy savings depend on the platform's compute-to-bandwidth balance: 24–32% on Orin versus 1–4% on Thor.*

4.3.3 Phase-Aware Frequency Policy.

Sections 4.3.1 and 4.3.2 show that the phases of one inference place different demands on host dispatch, memory bandwidth, and GPU compute. We derive three rules from these measurements and evaluate them together. *(i) CPU at maximum frequency.* Iterative action heads repeatedly dispatch small kernels and gain most from compilation and kernel fusion (Takeaway 1), so keeping the CPU at maximum frequency reduces host-side dispatch delay. *(ii) EMC at maximum frequency.* EMC throttling increases per-inference energy on both SoCs (Section 4.3.2), so we keep EMC at maximum frequency. *(iii) GPU frequency selected for each phase.* Phases with different resource demands respond differently to GPU clock (Takeaway 3), so we allow each phase to use a separate clock. Together, the fixed CPU and EMC settings and phase-specific GPU settings form the phase-aware policy evaluated below.

Sweep and selection. With the first two rules active, we sweep five candidate GPU clocks over the range used by each platform's governor: 306–1300 MHz on Orin and 315–1575 MHz on Thor. A two-phase model assigns one of the five clocks to each phase independently, so we evaluate all 25 clock combinations. The baseline is the default platform DVFS configuration, which leaves the GPU, EMC, and CPU frequencies under their default governors and enables CPU idle. We select the minimum-energy phase-wise assignment whose latency is lower than this baseline. Assignments that give both phases the same clock reduce to static whole-inference scaling, so we exclude them from selection. Figure 10 reports the sweep and the selected points.

Transition-free evaluation. The reported latency excludes measured clock-switch time. Phase-aware energy is recombined as $\sum_i t_i P_i$ from per-phase power measured at each static clock because the INA3221 cannot resolve the ~3.5 ms (Orin) and ~8.6 ms (Thor) transitions well enough to attribute their energy between phases. Figure 10 therefore measures the operating-point opportunity without transition cost. Section 4.3.4 quantifies the measured transition latency.

Models with dissimilar phases offer favorable operating points under the phase-aware policy. GR00T N1.6 and $\pi_{0.5}$ combine a compute-heavy backbone with an iteration-heavy action head whose resource demands differ from those of the backbone. Figure 10 shows that, for both models on both SoCs, phase-wise assignments under the phase-aware policy reduce latency and energy relative to default DVFS in the transition-free evaluation. On Orin, $\pi_{0.5}$ reduces energy by 17.9% and latency by 2.0%. On Thor, GR00T N1.6 reduces energy by 7.5% and latency by 18.8%, while $\pi_{0.5}$ reduces energy by 6.0% and latency by 18.4%.

Switching needs distinct, long phases. Cosmos-Policy's VAE and DiT are both dominated by large-GEMM work. On Orin, no phase-wise assignment is faster than the baseline because lowering either clock increases latency. On Thor, the selected point is 1.5% faster and 1.9% lower energy, but the energy difference lies within the 2.2% measurement resolution for that configuration. OpenVLA-OFT presents a different limit: its action head is a single 0.6 ms MLP pass (Section 4.1), shorter than one frequency transition, so only the backbone is long enough for GPU-clock assignment.

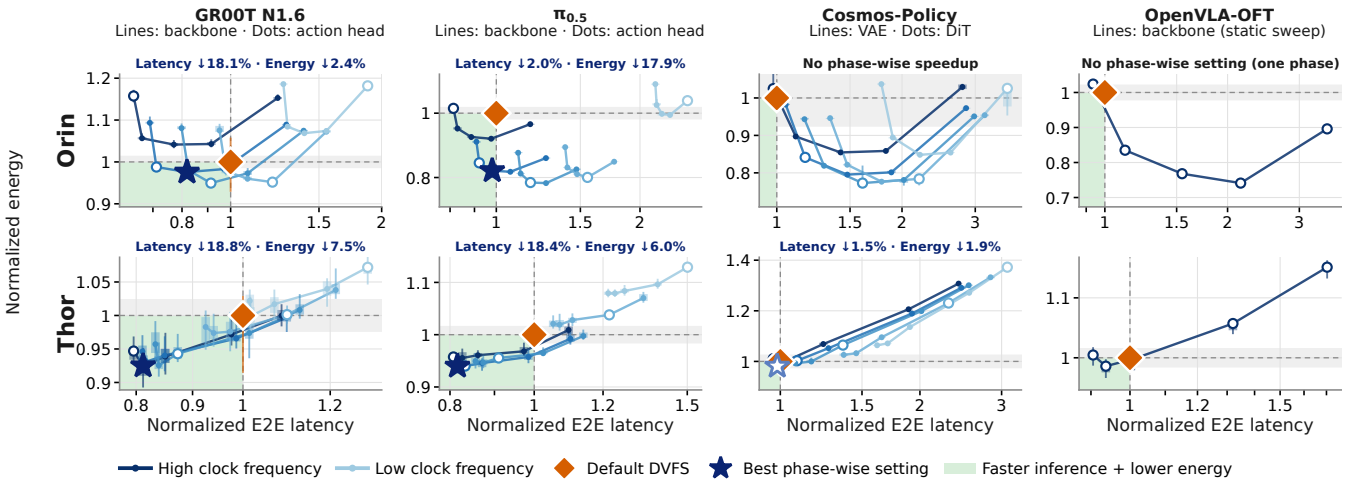


Figure 10. Transition-free latency–energy tradeoffs under the three-rule phase-aware policy on Orin and Thor, normalized to the default platform DVFS configuration (lower is better). Open stars denote energy changes within measurement resolution, and candlesticks show energy variability.

4.3.4 Transition Overhead.

Table 6 reports median transition overheads of 3.5 ms on Orin and 8.6 ms on Thor, and the resulting E2E share reaches 13.6% on Thor. We measure transition latency but do not separately attribute transition energy.

Transition latency limits gains from short phases. On Thor, GR00T N1.6 spends 13.6% of baseline E2E latency on two transitions, which consumes most of its 18.8% transition-free latency gain. Cosmos-Policy's longer inference spends less than 2% on the same two transitions. Phase heterogeneity determines the available operating-point gain, while phase duration relative to transition latency limits the realizable portion. We report this margin without implementing a runtime policy. Both axes of Figure 10 remain transition-free upper bounds.

The analysis thus far has treated each inference in isolation. In closed-loop deployment, per-inference latency governs overlap aggressiveness, shaping both control frequency and observation staleness. Section 5 quantifies this interaction.

Takeaway 5. *Phase-aware frequency control is useful when GPU phases differ in clock sensitivity and are long enough to amortize frequency transitions. GR00T N1.6 and $\pi_{0.5}$ meet these conditions, saving 2.4–17.9% energy while running faster when the clock-switch cost is excluded. The switching time itself is platform-dependent and can consume much of this latency gain when phases are short.*

5 Closed-Loop Serving Tradeoffs

This section maps single-inference costs to closed-loop behavior. We study asynchronous overlap (§5.1), accuracy–speed–energy tradeoffs (§5.2), throughput–energy efficiency (§5.3), and SLO-constrained operating points (§5.4).

Table 7. Worst-case round-trip time (RTT_{max}) and control periods per overlap level. RTT_{max} averages each episode's maximum RTT under synchronous operation and covers observation transfer, server-side pre- and post-processing, and compiled inference. Control periods are derived via Eq. 5.

			Control Period (ms)		
Model	**Platform**	**RTT_{max} (ms)**	**10%**	**25%**	**50%**
GR00T N1.6	RTX 4090	142.8	107.1	53.5	26.8
	Thor	126.7	95.1	47.5	23.8
	Orin	206.8	155.1	77.6	38.8
$\pi_{0.5}$	RTX 4090	136.1	204.1	68.0	40.8
	Thor	180.6	270.8	90.3	54.2
	Orin	457.3	686.0	228.7	137.2
OpenVLA-OFT	RTX 4090	148.7	223.0	111.5	55.8
	Thor	306.8	460.1	230.1	115.0
	Orin	528.2	792.2	396.1	198.1
Cosmos-Policy	RTX 4090	442.9	332.2	166.1	83.0
	Thor	1023.0	767.2	383.6	191.8
	Orin	2558.4	1918.8	959.4	479.7

5.1 Experimental Setup

Overlap mechanism and control period. A chunk contains K actions, and the controller executes one action per control period. The key design parameter of asynchronous overlap is the *overlap percentage* p, which determines when the next inference begins [6, 44, 46]. The system triggers it when $\lceil K \times p \rceil$ steps remain in the current chunk, leaving that many control periods for the inference to complete before a stall occurs. For K=16, p=10% provides two control periods for the next inference, whereas p=50% provides eight. The

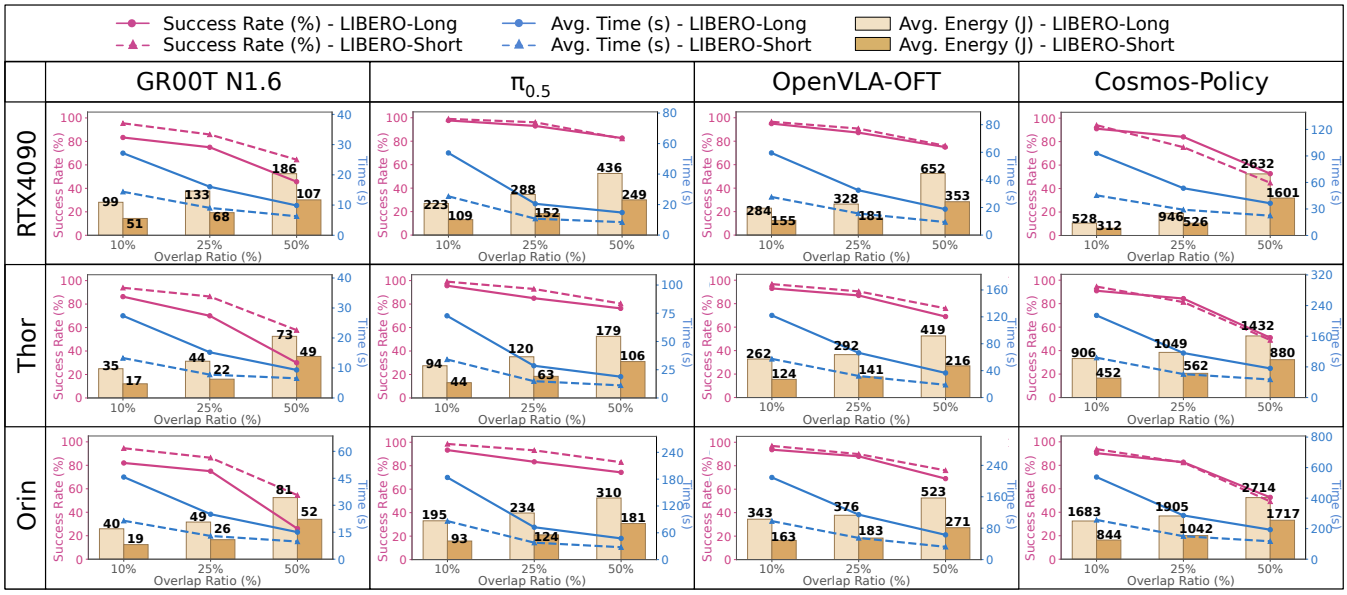


Figure 11. Accuracy–speed–energy tradeoff across all model–platform combinations, split into our two task groups: *Long* (LIBERO-10) and *Short* (LIBERO-Goal/Spatial/Object). Each subplot shows success rate (magenta, left axis), average episode time (blue, right axis), and average measured energy (tan bars, annotated in J) as a function of overlap percentage.

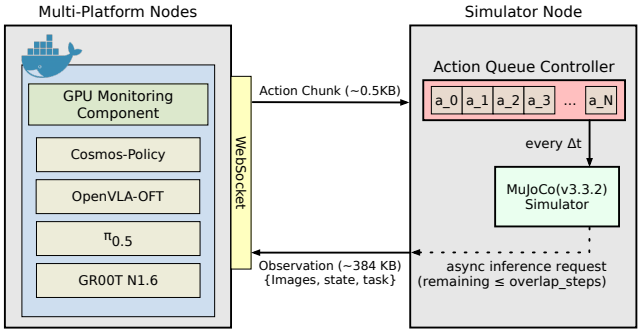


Figure 12. System architecture of the closed-loop benchmark. The client node runs a MuJoCo simulator and consumes actions from a FIFO queue at fixed intervals Δt, while asynchronously prefetching new action chunks from multi-platform model servers over WebSocket.

per-step control period must therefore satisfy

$$\Delta t \geq \frac{T_{\text{infer}}}{\lceil K \times p \rceil}, \tag{4}$$

where T_{infer} is inference latency. Higher p permits a shorter Δt but increases observation staleness (Eq. (1)), exposing the accuracy–staleness tradeoff of Section 2.2.

Serving configuration. For closed-loop evaluation, the RTX 4090 runs with default DVFS, while Jetson Thor and Jetson AGX Orin run at maximum clocks. We use the RTT measured under these settings to set the control period:

$$\Delta t = \frac{\text{RTT}_{\max} \times 1.5}{\lceil K \times p \rceil}, \tag{5}$$

where $\text{RTT}_{\max}$ is each episode's maximum round-trip time for one inference request, measured under synchronous operation and averaged across episodes, and the 1.5× factor adds headroom beyond the observed worst case. Compilation follows Table 5, per-configuration control periods are in Table 7, and the client–server system is shown in Figure 12.

Evaluation scale. We sweep $p \in \{10\%, 25\%, 50\%\}$ and run 30 independent rollouts per task across four models, three platforms, and four LIBERO suites (10 tasks each), grouped by horizon into a *Long* group (LIBERO-10) and a *Short* group (LIBERO-Goal, LIBERO-Spatial, LIBERO-Object), totaling 43,200 closed-loop episodes over 40 tasks.

5.2 The Accuracy–Speed–Energy Tradeoff

Figure 11 presents the three-way tradeoff across all model–platform combinations for the Long and Short task groups. Every configuration shows the same directional pattern as overlap grows from 10% to 50%: success rate decreases, time decreases, and energy increases. The rates, however, differ substantially.

Accuracy degradation is architecture-dependent. The success-rate drop from 10% to 50% overlap maintains a consistent ordering across platforms *and across both task groups*: $\pi_{0.5}$ degrades least, followed by OpenVLA-OFT, then GR00T N1.6 and Cosmos-Policy (on RTX 4090 the two robust architectures lose 15–20 pp, versus 31–49 pp for GR00T N1.6 and Cosmos-Policy).

Speed scaling is platform-invariant. For a fixed task group, the time reduction ratio from 10% to 50% is stable across platforms (e.g., GR00T N1.6 2.8–3.0×; $\pi_{0.5}$ 3.6–3.9×

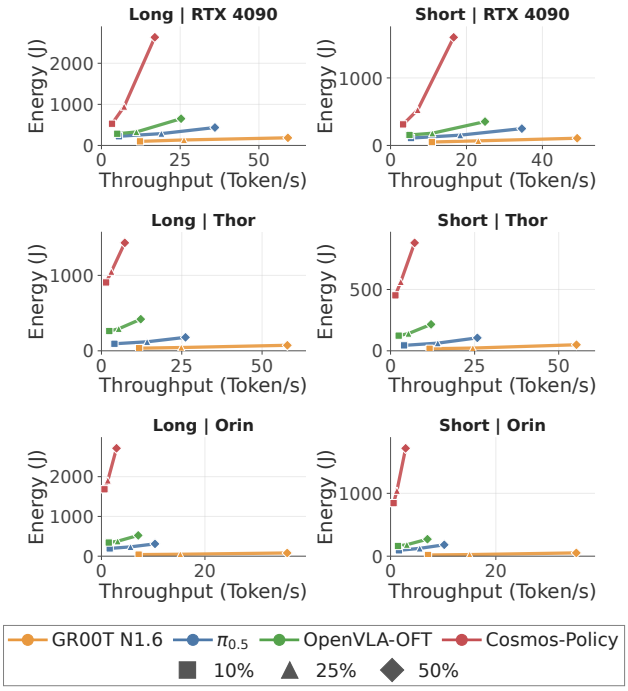


Figure 13. Task-equal successful-episode energy versus action throughput, per task group and platform, across overlap levels (10%–50%). Each trajectory connects one model's overlap settings; lower right (higher throughput at lower energy) is better.

on the Long group). This follows from the stage-proportion stability observed in Section 4.3: because the backbone-to-action-head latency ratio is preserved across platforms and control periods are calibrated to the measured worst-case RTT, the relative speedup is platform-independent.

Energy scaling is platform-dependent. Cosmos-Policy on RTX 4090 exhibits the steepest energy growth (up to 5.1× in the Short group from 10% to 50%), while GR00T N1.6 on the same platform increases only 1.9–2.1×. On edge SoCs, the range is 1.5–2.9× due to the lower power ceiling.

Takeaway 6. *Asynchronous overlap creates a three-way trade-off: accuracy degradation depends on architecture, relative speedup is stable across platforms, and energy scaling depends on platform.*

5.3 Throughput–Energy Efficiency

Figure 13 plots task-equal successful-episode energy against generated action throughput. Each point represents one model–platform–overlap combination. Each curve traces a model's trajectory as overlap increases from 10% to 50%, moving rightward (higher throughput) and upward (higher energy).

Memory-bound stage dominance corresponds to higher throughput-per-joule. GR00T N1.6 and $\pi_{0.5}$ trace shallow curves across all platforms. On RTX 4090, GR00T N1.6 reaches ~59 tokens/s at 186 J per successful episode; Cosmos-Policy reaches ~17 tokens/s at 2,632 J. On Thor, the same efficiency ordering holds: GR00T N1.6 achieves 58 tokens/s at 73 J, while Cosmos-Policy exceeds 1,000 J despite far lower throughput.

Edge platforms show a larger efficiency gap. The energy gap between the most and least efficient models spans roughly one order of magnitude on RTX 4090 and widens further on Thor. GR00T N1.6 and $\pi_{0.5}$ keep their shallow energy curves as throughput increases, whereas Cosmos-Policy and OpenVLA-OFT retain steep slopes. This ordering is consistent with the single-inference profiles in Section 4.2, where memory-bound stages, concentrated in the action heads, account for most of GR00T N1.6's and $\pi_{0.5}$'s GPU time, while compute-bound stages dominate Cosmos-Policy and OpenVLA-OFT.

Takeaway 7. *The stage-level bottleneck mix identified by single-inference profiling (Takeaway 2) is reflected in deployment efficiency. In the evaluated models, dominance by memory-bound action heads corresponds to over an order of magnitude higher throughput-per-joule, and edge platforms widen this gap.*

5.4 Deployment Operating Points Under SLO Constraints

Figure 14 presents operating points under four resource and timing constraints. We additionally evaluate an SR-first minimum-energy objective. To apply a common selection rule, we set each budget to the lower quartile (p25) of its measured configuration distribution and select the highest-SR feasible point.

$\pi_{0.5}$ is selected under all four p25 budgets, while the constraint determines its platform and overlap. The 15.6 s spend-time budget selects RTX 4090 at 50% overlap on Long (82.7% SR) and 25% on Short (96.1% SR). The 75.2 ms control-cycle budget also selects RTX 4090, at 25% overlap for both groups (93.0% and 96.1% SR). The edge-only 2.1 W power budget and the 98.0 J energy budget select Thor at 10% overlap for both groups (95.7% and 99.1% SR). Under the energy budget, the Long and Short selections consume 93.5 J and 44.4 J, respectively.

The selected model changes with the objective: from highest SR within a resource budget to minimum energy above an SR threshold. With an 85% SR threshold, GR00T N1.6 on Thor at 10% is selected for both Long (86.3%, 34.8 J) and Short (93.9%, 16.8 J), although the Long result is close to the threshold. OpenVLA-OFT and Cosmos-Policy are not selected under either objective. These selections do not establish model-level dominance: the budgets and the threshold are analysis choices, and different cut-offs could select different configurations.

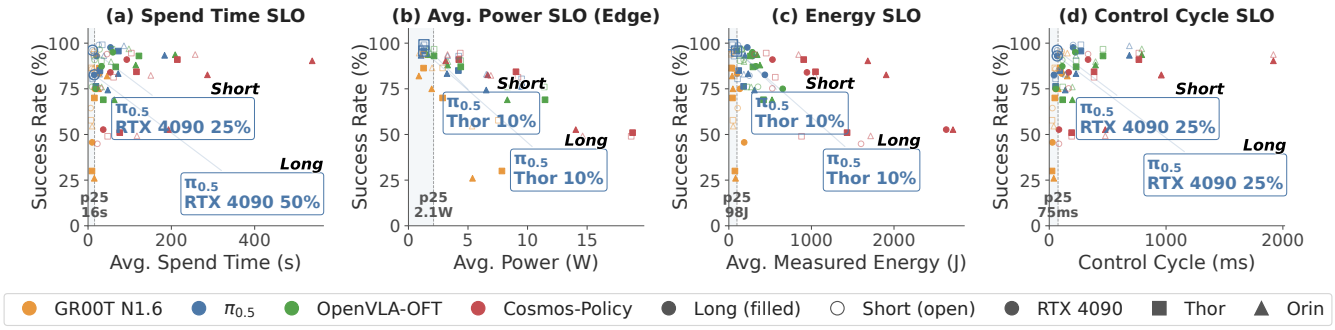


Figure 14. Deployment operating points under SLO constraints. Shading marks values below each metric's lower-quartile (p25) budget; callouts mark the highest-SR feasible Long and Short points. Filled markers: Long (LIBERO-10); open markers: Short (LIBERO-Goal/Spatial/Object).

Takeaway 8. *No single model–platform–overlap configuration is Pareto-dominant across time, energy, power, and control-cycle constraints. The optimal operating point shifts with the SLO type and task group, though $\pi_{0.5}$ is selected under most SLOs.*

6 Discussion

Staleness-aware VLA serving abstraction. Our measurements motivate a runtime interface that exposes four kinds of state: per-stage cost, action-chunk progress, observation freshness, and deployment SLOs. Concretely, the interface reports each stage's latency and compute/bandwidth demand, the number of unexecuted actions in the current chunk and the time at which they run out, the timestamp and staleness budget of the latest observation, and the target SLOs for latency, energy, power, and task success. A scheduler could use this state to select execution timing, overlap ratio, resource settings, and where each stage executes. LLM prefill–decode disaggregation schedules worker pools across concurrent requests [63]. A batch-1 VLA loop instead schedules stages within one robot's inference, where the execution location of a stage changes observation staleness and task success in addition to latency (Section 5).

Architecture implications. The measured bottlenecks suggest several hardware features that could improve VLA inference, although we do not propose a specific accelerator design. Large GEMMs in backbones and video-generation DiTs benefit from high tensor-compute throughput. Denoising loops instead issue many small, fixed-shape GEMMs, so they benefit from hardware that sustains high utilization on small matrix operations. Memory-bound action heads benefit from higher memory bandwidth, and iteration-heavy action heads benefit from low kernel-launch and control overhead. A denoising loop also reloads the same weights at every step, so cache or local memory that keeps them on chip could reduce this repeated traffic. Finally, faster frequency transitions would reduce the measured overhead that limits our phase-aware DVFS policy.

Selective applicability of existing optimizations. The control-period deadline favors on-board or nearby edge execution because remote execution adds network latency and jitter to the observation-staleness budget (Section 5). In the single-robot setting studied here, requests arrive one at a time, so inference runs at batch size one. We assess the optimizations discussed below using three workload properties: request concurrency, autoregressive generation, and the dominant execution stage.

Techniques that optimize computation within a single inference (kernel fusion and graph capture, step distillation, feature caching, weight-only quantization) remain applicable at batch size one, although their benefit depends on the dominant stage. Among these, we directly evaluate compilation, which delivers 4.4–7.4× speedups on iteration-heavy action heads (Section 4.1.2).

Request-level and KV-cache optimizations are more constrained in this setting. Because a single robot issues only one request at a time, continuous batching offers little benefit unless a server handles requests from multiple robots. The autoregressive baseline still uses a KV cache, but its decode length is short and fixed by the action format, so it is known before decoding starts. This leaves little opportunity for paged KV-cache management to reduce fragmentation or improve memory utilization. The cache is also discarded after each action is generated, so the profiled implementation retains no prefix to reuse across inferences (Section 2.1).

Techniques specific to autoregressive decoding, such as speculative decoding, have a still narrower scope. Among the models we study, they apply only to the autoregressive baseline. Even there, the decode is so short that the overhead of running a draft model may outweigh the benefit. Except for compilation, these conclusions are based on workload structure rather than measured speedups.

Scope and limitations. *System-design scope.* We characterize the constraints a VLA runtime must handle, but we do not implement a scheduler. A deployable phase-aware policy must account for transition overhead and phase duration.

Stage co-execution must account for model dependencies and shared-memory contention.

Simulation scope. Across 40 LIBERO tasks, we compare the models under identical MuJoCo dynamics in 43,200 episodes spanning model, platform, and overlap configurations. MuJoCo provides controller-mediated actuation dynamics, inference timing comes from the target hardware, and each control period includes a conservative 1.5× margin over the measured worst-case RTT. This setup relates control frequency and observation staleness to task success under controlled conditions. It does not replace real-robot or hardware-in-the-loop validation: sensor pipelines, network jitter, multi-camera synchronization, unmodeled contact dynamics, and physical disturbances are outside our scope.

Model scope. We profile autoregressive OpenVLA as the discrete action-token baseline. The closed-loop evaluation covers the four action-chunk models used for continuous high-rate control. The OpenVLA profile retains the known memory-bound LLM decode behavior. Alternative autoregressive control interfaces are outside our scope.

Takeaway 2 may extend to other models in the same paradigm, although exact stage proportions depend on model size and chunk length. Our model set excludes 1-step distillation variants, which can remove or reduce the memory-bound action-head stage. Whether the accuracy–staleness ordering (Takeaway 6) holds for such architectures remains open.

Hardware scope. We evaluate one discrete edge server and two unified-memory onboard SoCs, all from NVIDIA. Sensitivity varies across them with compute-to-bandwidth balance. Non-NVIDIA accelerators are not evaluated.

Energy-measurement scope. Cross-platform energy comparisons reflect relative trends rather than absolute ratios due to differing measurement boundaries (Section 3), while within-platform comparisons are directly comparable.

7 Related Work

Algorithmic perspectives on VLA efficiency. Recent surveys organize the VLA landscape along algorithmic axes: Kawaharazuka et al. [20] provide a full-stack review spanning architectures and robot platforms, Yu et al. [56] propose a taxonomy of efficient VLAs across model design, training, and data collection, and Guan et al. [15] categorize efficiency techniques by architecture, perception, action generation, and training/inference strategy. These works describe proposed methods but provide limited empirical characterization of how deployed systems behave on specific hardware. Deployment-level bottleneck attribution, DVFS-sensitive energy behavior, and closed-loop operating-point analysis receive less empirical attention.

LLM serving systems. LLM servers improve throughput and goodput for concurrent autoregressive requests through scheduling and batching [1, 55], paged KV-cache management [24], and prefill–decode disaggregation [39, 63]. VLA serving changes both the scheduling unit and the objective. A single-robot batch-1 loop schedules backbone and action-head stages instead of prefill and decode phases across independent requests. Placement and queuing delay also increase observation staleness, coupling task success to control-period, accuracy, and energy constraints.

ML inference characterization. Roofline models explain memory-bound LLM decoding and guide serving optimization [57]. RooflineBench [3] extends operational-intensity comparison to on-device LLMs. Roofline analysis also explains the memory-bound autoregressive OpenVLA decode loop in Section 2.1. We apply it to heterogeneous action-chunk computation across iterative denoising, parallel decoding, and video generation, and separately measure closed-loop staleness in batch-1 serving.

8 Conclusion

This work presents a system-level workload characterization of embodied action generation model serving across four models, three platforms, and 40 LIBERO tasks. At the single-inference level, we show that the compute- versus memory-bound behavior is largely governed by action tensor dimensionality. Under closed-loop overlap, accuracy degradation depends on model architecture, while energy scaling depends on the hardware platform, with no configuration remaining Pareto-dominant across SLOs. These results highlight the need for architecture-aware resource allocation and staleness-aware operating-point selection. Together, our findings call for VLA systems that jointly consider model architecture, hardware resource balance, and runtime policies, providing a foundation for future accelerators and serving systems tailored to embodied AI.

Acknowledgments

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) funded by the Ministry of Science and ICT, Korea (RS-2024-00396013 and RS-2024-00402898), and the National Research Foundation of Korea (NRF; RS-2024-00347114).

References

[1] Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. 2024. Taming Throughput-Latency Tradeoff in LLM Inference with Sarathi-Serve. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/agrawal

[2] Lucas Beyer, Andreas Steiner, André Susano Pinto, Alexander Kolesnikov, Xiao Wang, Daniel Salz, Maxim Neumann, Ibrahim Alabdulmohsin, Michael Tschannen, Emanuele Bugliarello, Thomas Unterthiner, Daniel Keysers, Skanda Koppula, Fangyu Liu, Adam Grycner, Alexey A. Gritsenko, Neil Houlsby, Manoj Kumar, Keran Rong, Julian Eisenschlos, Rishabh Kabra, Matthias Bauer, Matko Bosnjak, Xi Chen, Matthias Minderer, Paul Voigtlaender, Ioana Bica, Ivana Balazevic, Joan Puigcerver, Pinelopi Papalampidi, Olivier J. Hénaff, Xi Xiong, Radu Soricut, Jeremiah Harmsen, and Xiaohua Zhai. 2024. PaliGemma: A

versatile 3B VLM for transfer. *arXiv preprint arXiv:2407.07726* (2024). doi:10.48550/arXiv.2407.07726

[3] Zhen Bi, Xueshu Chen, Luoyang Sun, Yuhang Yao, Qing Shen, Jungang Lou, and Cheng Deng. 2026. RooflineBench: A Benchmarking Framework for On-Device LLMs via Roofline Analysis. *arXiv preprint arXiv:2602.11506* (2026). doi:10.48550/arXiv.2602.11506

[4] Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, and Ury Zhilinsky. 2025. $\pi_{0.5}$: a Vision-Language-Action Model with Open-World Generalization. In *Conference on Robot Learning (CoRL)*. https://proceedings.mlr.press/v305/black25a.html

[5] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Robert Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, Laura Smith, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. 2025. π_0: A Vision-Language-Action Flow Model for General Robot Control. In *Robotics: Science and Systems (RSS)*. doi:10.15607/RSS.2025.XXI.010

[6] Kevin Black, Manuel Galliker, and Sergey Levine. 2025. Real-Time Execution of Action Chunking Flow Policies. In *Advances in Neural Information Processing Systems (NeurIPS)*. doi:10.52202/085713-1122

[7] Qingwen Bu, Hongyang Li, Li Chen, Jisong Cai, Jia Zeng, Heming Cui, Maoqing Yao, and Yu Qiao. 2024. Towards Synergistic, Generalized, and Efficient Dual-System for Robotic Manipulation. *arXiv preprint arXiv:2410.08001* (2024). doi:10.48550/arXiv.2410.08001

[8] Chilam Cheang, Sijin Chen, Zhongren Cui, Yingdong Hu, Liqun Huang, Tao Kong, Hang Li, Yifeng Li, Yuxiao Liu, Xiao Ma, Hao Niu, Wenxuan Ou, Wanli Peng, Zeyu Ren, Haixin Shi, Jiawen Tian, Hongtao Wu, Xin Xiao, Yuyang Xiao, Jiafeng Xu, and Yichu Yang. 2025. GR-3 Technical Report. *arXiv preprint arXiv:2507.15493* (2025). doi:10.48550/arXiv.2507.15493

[9] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. 2014. cuDNN: Efficient Primitives for Deep Learning. *arXiv preprint arXiv:1410.0759* (2014). doi:10.48550/arXiv.1410.0759

[10] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. 2025. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research (IJRR)* 44, 10–11 (2025), 1684–1704. doi:10.1177/02783649241273668

[11] Shengliang Deng, Mi Yan, Songlin Wei, Haixin Ma, Yuxin Yang, Jiayi Chen, Zhiqi Zhang, Taoyu Yang, Xuheng Zhang, Heming Cui, Zhizheng Zhang, and He Wang. 2025. GraspVLA: a Grasping Foundation Model Pre-trained on Billion-scale Synthetic Action Data. In *Conference on Robot Learning (CoRL)*. https://proceedings.mlr.press/v305/deng25a.html

[12] Nan Ding and Samuel Williams. 2019. An Instruction Roofline Model for GPUs. In *Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. doi:10.1109/PMBS49563.2019.00007

[13] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=YicbFdNTTy

[14] Dibya Ghosh, Homer Rich Walke, Karl Pertsch, Kevin Black, Oier Mees, Sudeep Dasari, Joey Hejna, Tobias Kreiman, Charles Xu, Jianlan Luo, You Liang Tan, Lawrence Yunliang Chen, Quan Vuong, Ted Xiao, Pannag R Sanketi, Dorsa Sadigh, Chelsea Finn, and Sergey Levine. 2024. Octo: An Open-Source Generalist Robot Policy. In *Robotics: Science and Systems (RSS)*. doi:10.15607/RSS.2024.XX.090

[15] Weifan Guan, Qinghao Hu, Aosheng Li, and Jian Cheng. 2025. Efficient Vision-Language-Action Models for Embodied Manipulation: A Systematic Survey. *arXiv preprint arXiv:2510.17111* (2025). doi:10.48550/arXiv.2510.17111

[16] Yanjiang Guo, Yucheng Hu, Jianke Zhang, Yen-Jen Wang, Xiaoyu Chen, Chaochao Lu, and Jianyu Chen. 2024. Prediction with Action: Visual Policy Learning via Joint Denoising Process. In *Advances in Neural Information Processing Systems (NeurIPS)*. doi:10.52202/079017-3570

[17] Mark Hill and Vijay Janapa Reddi. 2019. Gables: A Roofline Model for Mobile SoCs. In *International Symposium on High Performance Computer Architecture (HPCA)*. doi:10.1109/HPCA.2019.00047

[18] Jonathan Ho, Ajay Jain, and Pieter Abbeel. 2020. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems (NeurIPS)*. https://proceedings.neurips.cc/paper_files/paper/2020/file/4c5bcfec8584af0d967f1ab10179ca4b-Paper.pdf

[19] Dong Jing, Gang Wang, Jiaqi Liu, Weiliang Tang, Zelong Sun, Yunchao Yao, Zhenyu Wei, Yunhui Liu, Zhiwu Lu, and Mingyu Ding. 2026. Mixture of Horizons in Action Chunking. In *International Conference on Machine Learning (ICML)*. https://proceedings.mlr.press/v306/jing26f.html

[20] Kento Kawaharazuka, Jihoon Oh, Jun Yamada, Ingmar Posner, and Yuke Zhu. 2025. Vision-Language-Action Models for Robotics: A Review Towards Real-World Applications. *IEEE Access* 13 (2025), 162467–162504. doi:10.1109/ACCESS.2025.3609980

[21] Moo Jin Kim, Chelsea Finn, and Percy Liang. 2025. Fine-Tuning Vision-Language-Action Models: Optimizing Speed and Success. In *Robotics: Science and Systems (RSS)*. doi:10.15607/RSS.2025.XXI.017

[22] Moo Jin Kim, Yihuai Gao, Tsung-Yi Lin, Yen-Chen Lin, Yunhao Ge, Grace Lam, Percy Liang, Shuran Song, Ming-Yu Liu, Chelsea Finn, and Jinwei Gu. 2026. Cosmos Policy: Fine-Tuning Video Models for Visuomotor Control and Planning. In *International Conference on Learning Representations (ICLR)*. https://proceedings.iclr.cc/paper_files/paper/2026/file/748becc400a57c0e31cfe6a2e7951467-Paper-Conference.pdf

[23] Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. 2025. OpenVLA: An Open-Source Vision-Language-Action Model. In *Conference on Robot Learning (CoRL)*. https://proceedings.mlr.press/v270/kim25c.html

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Symposium on Operating Systems Principles (SOSP)*. doi:10.1145/3600006.3613165

[25] Qixiu Li, Yaobo Liang, Zeyu Wang, Lin Luo, Xi Chen, Mozheng Liao, Fangyun Wei, Yu Deng, Sicheng Xu, Yizhong Zhang, Xiaofan Wang, Bei Liu, Jianlong Fu, Jianmin Bao, Dong Chen, Yuanchun Shi, Jiaolong Yang, and Baining Guo. 2024. CogACT: A Foundational Vision-Language-Action Model for Synergizing Cognition and Action in Robotic Manipulation. *arXiv preprint arXiv:2411.19650* (2024). doi:10.48550/arXiv.2411.19650

[26] Shuang Li, Yihuai Gao, Dorsa Sadigh, and Shuran Song. 2025. Unified Video Action Model. In *Robotics: Science and Systems (RSS)*. doi:10.15607/RSS.2025.XXI.074

[27] Wei Li, Renshan Zhang, Rui Shao, Jie He, and Liqiang Nie. 2025. CogVLA: Cognition-Aligned Vision-Language-Action Models via Instruction-Driven Routing & Sparsification. In *Advances in Neural Information Processing Systems (NeurIPS)*. doi:10.52202/085713-4594

[28] Zhiqi Li, Guo Chen, Shilong Liu, Shihao Wang, Vibashan VS, Yishen Ji, Shiyi Lan, Hao Zhang, Yilin Zhao, Subhashree Radhakrishnan, Nadine Chang, Karan Sapra, Amala Sanjay Deshmukh, Tuomas Rintamaki, Matthieu Le, Ilia Karmanov, Lukas Voegtle, Philipp Fischer, De-An Huang, Timo Roman, Tong Lu, José M. Álvarez, Bryan Catanzaro, Jan Kautz, Andrew Tao, Guilin Liu, and Zhiding Yu. 2025. Eagle 2: Building Post-Training Data Strategies from Scratch for Frontier Vision-Language Models. *arXiv preprint arXiv:2501.14818* (2025). doi:10.48550/arXiv.2501.14818

[29] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. 2023. Flow Matching for Generative Modeling. In *International Conference on Learning Representations (ICLR)*. https://openreview.net/forum?id=PqvMRDCJT9t

[30] Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. 2023. LIBERO: Benchmarking Knowledge Transfer for Lifelong Robot Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*. doi:10.52202/075280-1939

[31] Yueen Ma, Zixing Song, Yuzheng Zhuang, Jianye Hao, and Irwin King. 2026. A Survey on Vision–Language–Action Models for Embodied AI. *IEEE Transactions on Neural Networks and Learning Systems (TNNLS)* 37, 7 (2026), 3031–3051. doi:10.1109/TNNLS.2025.3650584

[32] NVIDIA, Niket Agarwal, Arslan Ali, Maciej Bala, Yogesh Balaji, Erik Barker, Tiffany Cai, Prithvijit Chattopadhyay, Yongxin Chen, Yin Cui, Yifan Ding, Daniel Dworakowski, Jiaojiao Fan, Michele Fenzi, Francesco Ferroni, Sanja Fidler, Dieter Fox, Songwei Ge, Yunhao Ge, Jinwei Gu, Siddharth Gururani, Ethan He, Jiahui Huang, Jacob Huffman, Pooya Jannaty, Jingyi Jin, Seung Wook Kim, Gergely Klár, Grace Lam, Shiyi Lan, Laura Leal-Taixe, Anqi Li, Zhaoshuo Li, Chen-Hsuan Lin, Tsung-Yi Lin, Huan Ling, Ming-Yu Liu, Xian Liu, Alice Luo, Qianli Ma, Hanzi Mao, Kaichun Mo, Arsalan Mousavian, Seungjun Nah, Sriharsha Niverty, David Page, Despoina Paschalidou, Zeeshan Patel, Lindsey Pavao, Morteza Ramezanali, Fitsum Reda, Xiaowei Ren, Vasanth Rao Naik Sabavat, Ed Schmerling, Stella Shi, Bartosz Stefaniak, Shitao Tang, Lyne Tchapmi, Przemek Tredak, Wei-Cheng Tseng, Jibin Varghese, Hao Wang, Haoxiang Wang, Heng Wang, Ting-Chun Wang, Fangyin Wei, Xinyue Wei, Jay Zhangjie Wu, Jiashu Xu, Wei Yang, Lin Yen-Chen, Xiaohui Zeng, Yu Zeng, Jing Zhang, Qinsheng Zhang, Yuxuan Zhang, Qingqing Zhao, and Artur Zolkowski. 2025. Cosmos World Foundation Model Platform for Physical AI. *arXiv preprint arXiv:2501.03575* (2025). doi:10.48550/arXiv.2501.03575

[33] NVIDIA, Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. 2025. GR00T N1: An Open Foundation Model for Generalist Humanoid Robots. *arXiv preprint arXiv:2503.14734* (2025). doi:10.48550/arXiv.2503.14734

[34] NVIDIA Corporation. 2025. CUDA Programming Guide. https://docs.nvidia.com/cuda/archive/13.1.0/cuda-programming-guide/index.html Version 13.1. Accessed: 2026-09-22.

[35] NVIDIA Corporation. 2025. Platform Power and Performance: Jetson Orin Nano Series, Jetson Orin NX Series and Jetson AGX Orin Series. https://docs.nvidia.com/jetson/archives/r36.4.3/DeveloperGuide/SD/PlatformPowerAndPerformance/JetsonOrinNanoSeriesJetsonOrinNxSeriesAndJetsonAgxOrinSeries.html. Accessed: 2026-09-22.

[36] NVIDIA Corporation. 2025. Platform Power and Performance: Jetson Thor Product Family. https://docs.nvidia.com/jetson/archives/r38.2/DeveloperGuide/SD/PlatformPowerAndPerformance/JetsonThor.html. Accessed: 2026-09-22.

[37] NVIDIA Corporation. 2026. NVIDIA Jetson Modules. https://developer.nvidia.com/embedded/jetson-modules. Accessed: 2026-09-22.

[38] NVIDIA Corporation. 2026. TensorRT: Architecture Overview. https://docs.nvidia.com/deeplearning/tensorrt/developer-guide/. Accessed: 2026-09-22.

[39] Pratyush Patel, Esha Choukse, Chaojie Zhang, Aashaka Shah, Íñigo Goiri, Saeed Maleki, and Ricardo Bianchini. 2024. Splitwise: Efficient Generative LLM Inference Using Phase Splitting. In *International Symposium on Computer Architecture (ISCA)*. doi:10.1109/ISCA59077.2024.00019

[40] William Peebles and Saining Xie. 2023. Scalable Diffusion Models with Transformers. In *International Conference on Computer Vision (ICCV)*. doi:10.1109/ICCV51070.2023.00387

[41] Padmanabhan Pillai and Kang G. Shin. 2001. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Symposium on Operating Systems Principles (SOSP)*. doi:10.1145/502034.502044

[42] Ranjan Sapkota, Yang Cao, Konstantinos I. Roumeliotis, and Manoj Karkee. 2025. Vision-Language-Action Models: Concepts, Progress, Applications and Challenges. *arXiv preprint arXiv:2505.04769* (2025). doi:10.48550/arXiv.2505.04769

[43] Kohei Sendai, Maxime Alvarez, Tatsuya Matsushima, Yutaka Matsuo, and Yusuke Iwasawa. 2025. Leave No Observation Behind: Real-time Correction for VLA Action Chunks. *arXiv preprint arXiv:2509.23224* (2025). doi:10.48550/arXiv.2509.23224

[44] Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andrés Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Rémi Cadène. 2025. SmolVLA: A Vision-Language-Action Model for Affordable and Efficient Robotics. *arXiv preprint arXiv:2506.01844* (2025). doi:10.48550/arXiv.2506.01844

[45] Wenxuan Song, Jiayi Chen, Pengxiang Ding, Han Zhao, Wei Zhao, Zhide Zhong, Zongyuan Ge, Zhijun Li, Donglin Wang, Lujia Wang, Jun Ma, and Haoang Li. 2025. PD-VLA: Accelerating Vision-Language-Action Model Integrated with Action Chunking via Parallel Decoding. In *International Conference on Intelligent Robots and Systems (IROS)*. doi:10.1109/IROS60139.2025.11247519

[46] Jiaming Tang, Yufei Sun, Yilong Zhao, Shang Yang, Yujun Lin, Zhuoyang Zhang, James Hou, Yao Lu, Zhijian Liu, and Song Han. 2025. VLASH: Real-Time VLAs via Future-State-Aware Asynchronous Inference. *arXiv preprint arXiv:2512.01031* (2025). doi:10.48550/arXiv.2512.01031

[47] Chunlin Tian, Xinpeng Qin, Kahou Tam, Li Li, Zijian Wang, Yuanzhe Zhao, Minglei Zhang, and Chengzhong Xu. 2025. CLONE: Customizing LLMs for Efficient Latency-Aware Inference at the Edge. In *USENIX Annual Technical Conference (USENIX ATC)*. https://www.usenix.org/conference/atc25/presentation/tian

[48] Emanuel Todorov, Tom Erez, and Yuval Tassa. 2012. MuJoCo: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems (IROS)*. doi:10.1109/IROS.2012.6386109

[49] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open Foundation

and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288* (2023). doi:10.48550/arXiv.2307.09288

[50] Haoxuan Wang, Gengyu Zhang, Yan Yan, Yuzhang Shang, Ramana Kompella, and Gaowen Liu. 2026. Real-Time Robot Execution with Masked Action Chunking. In *International Conference on Learning Representations (ICLR)*. https://proceedings.iclr.cc/paper_files/paper/2026/file/c35834443b7881e782e52b3519fe27c7-Paper-Conference.pdf

[51] Zibo Wang, Yijia Zhang, Fuchun Wei, Bingqiang Wang, Yanlin Liu, Zhiheng Hu, Jingyi Zhang, Xiaoxin Xu, Jian He, Xiaoliang Wang, Wanchun Dou, Guihai Chen, and Chen Tian. 2025. Using Analytical Performance/Power Model and Fine-Grained DVFS to Enhance AI Accelerator Energy Efficiency. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. doi:10.1145/3669940.3707231

[52] Junjie Wen, Yichen Zhu, Minjie Zhu, Zhibin Tang, Jinming Li, Zhongyi Zhou, Xiaoyu Liu, Chaomin Shen, Yaxin Peng, and Feifei Feng. 2025. DiffusionVLA: Scaling Robot Foundation Models via Unified Diffusion and Autoregression. In *International Conference on Machine Learning (ICML)*. https://proceedings.mlr.press/v267/wen25g.html

[53] Samuel Williams, Andrew Waterman, and David A. Patterson. 2009. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM (CACM)* 52, 4 (2009), 65–76. doi:10.1145/1498765.1498785

[54] Zebin Yang, Yijiahao Qi, Tong Xie, Bo Yu, Shaoshan Liu, and Meng Li. 2026. DySL-VLA: Efficient Vision-Language-Action Model Inference via Dynamic-Static Layer-Skipping for Robot Manipulation. *arXiv preprint arXiv:2602.22896* (2026). doi:10.48550/arXiv.2602.22896

[55] Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. 2022. Orca: A Distributed Serving System for Transformer-Based Generative Models. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi22/presentation/yu

[56] Zhaoshu Yu, Bo Wang, Pengpeng Zeng, Haonan Zhang, Ji Zhang, Zheng Wang, Lianli Gao, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2026. A Survey on Efficient Vision-Language-Action Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2026). doi:10.1109/TPAMI.2026.3734740 Early access.

[57] Zhihang Yuan, Yuzhang Shang, Yang Zhou, Zhen Dong, Zhe Zhou, Chenhao Xue, Bingzhe Wu, Zhikai Li, Qingyi Gu, Yong Jae Lee, Yan Yan, Beidi Chen, Guangyu Sun, and Kurt Keutzer. 2024. LLM Inference Unveiled: Survey and Roofline Model Insights. *arXiv preprint arXiv:2402.16363* (2024). doi:10.48550/arXiv.2402.16363

[58] Xiaohua Zhai, Basil Mustafa, Alexander Kolesnikov, and Lucas Beyer. 2023. Sigmoid Loss for Language Image Pre-Training. In *International Conference on Computer Vision (ICCV)*. doi:10.1109/ICCV51070.2023.01100

[59] Tony Z. Zhao, Vikash Kumar, Sergey Levine, and Chelsea Finn. 2023. Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware. In *Robotics: Science and Systems (RSS)*. doi:10.15607/RSS.2023.XIX.016

[60] Jinliang Zheng, Jianxiong Li, Dongxiu Liu, Yinan Zheng, Zhihao Wang, Zhonghong Ou, Yu Liu, Jingjing Liu, Ya-Qin Zhang, and Xianyuan Zhan. 2025. Universal Actions for Enhanced Embodied Foundation Models. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. doi:10.1109/CVPR52734.2025.02096

[61] Zihao Zheng, Hangyu Cao, Jiayu Chen, Sicheng Tian, Chenyue Li, Maoliang Li, Xinhao Sun, Guojie Luo, and Xiang Chen. 2026. RoboECC: Multi-Factor-Aware Edge-Cloud Collaborative Deployment for VLA Models. *arXiv preprint arXiv:2603.20711* (2026). doi:10.48550/arXiv.2603.20711

[62] Zhen Zheng, Xin Ji, Taosong Fang, Fanghao Zhou, Chuanjie Liu, and Gang Peng. 2026. BatchLLM: Optimizing Large Batched LLM Inference with Global Prefix Sharing and Throughput-oriented Token Batching. In *Proceedings of Machine Learning and Systems (MLSys)*. https://proceedings.mlsys.org/paper_files/paper/2026/file/5b7ae1758452854dee4e962207d38304-Paper-Conference.pdf

[63] Yinmin Zhong, Shengyu Liu, Junda Chen, Jianbo Hu, Yibo Zhu, Xuanzhe Liu, Xin Jin, and Hao Zhang. 2024. DistServe: Disaggregating Prefill and Decoding for Goodput-optimized Large Language Model Serving. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. https://www.usenix.org/conference/osdi24/presentation/zhong-yinmin

[64] Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. 2023. RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control. In *Conference on Robot Learning (CoRL)*. https://proceedings.mlr.press/v229/zitkovich23a.html