VLA-DUO: PERIODIC CROSS-INFERENCE SCHEDULING FOR MOBILE VISION-LANGUAGE-ACTION INFERENCE

ABSTRACT

Mobile SoCs are an attractive platform for onboard vision-language-action (VLA) inference in robots with tight space and power constraints. However, their limited compute capacity makes it difficult to generate action chunks fast enough to support high-frequency control. We present VLA-Duo, a periodic GPU–NPU scheduling system that overlaps successive VLA inferences to increase action-chunk throughput at the cost of higher observation-to-action latency. For a fixed task configuration, successive VLA inferences use the same operator sequence and tensor shapes, allowing the same execution plan to be repeated. We generate the plan offline from GPU–NPU performance profiles, accounting for inter-accelerator interference. At runtime, the scheduler follows the plan to coordinate the two accelerators and overlap successive inferences. Short-run evaluation with π_0 and $\pi_{0.5}$ on Snapdragon 8 Elite and 8 Elite Gen 5 shows that our system achieves 36.1–78.3% higher throughput than the ExecuTorch/QNN NPU, our strongest external NPU baseline. Output periods of 394–499 ms support action throughput equivalent to 10.0–12.7 Hz control when five actions are executed per chunk. In separate LIBERO simulations, $\pi_{0.5}$ achieves 94.5% average success under a ten-control-step observation-to-action delay using prefix inpainting without additional training or gradient computation.

1 INTRODUCTION

Vision-language-action (VLA) models map visual observations and language instructions to robot actions Black et al. (2025a). Deploying these models in products such as robots and autonomous vehicles requires local computation within practical power, space, and cost constraints. A desktop GPU provides ample compute for inference, while a mobile SoC meets these constraints and carries two accelerators, a GPU and an NPU, in one compact package.

The compute available on such a package sets how often the robot receives a new action chunk. In our measurements, a single $\pi_{0.5}$ inference using llama.cpp Gerganov & llama.cpp contributors (2026) on Snapdragon 8 Elite Gen 5 takes 4.36 s on the GPU and 2.67 s on the NPU. ExecuTorch with the QNN backend Nachin et al. (2026), the fastest runtime we measured, brings this interval to 550–889 ms across π_0, $\pi_{0.5}$, and two SoC generations (Table 2). With five actions used per chunk, that interval corresponds to a supported action rate of 5.6–9.1 Hz.

Inference optimization shortens this interval through efficient kernels and heterogeneous execution Gerganov & llama.cpp contributors (2026); Chen et al. (2025), while asynchronous control, as in RTC Black et al. (2025b) and VLASH Tang et al. (2025), overlaps action execution with inference so that the robot keeps moving between chunks. We extend this overlap to the computation itself: while one accelerator continues an inference, the other begins processing a newer observation. Adjacent inferences then occupy the two accelerators at the same time, and action chunks arrive more often than one accelerator alone produces them.

Three properties of VLA inference on mobile SoCs make this overlap practical. Under a fixed task configuration, successive inferences repeat the same operator sequence and tensor shapes, so an execution plan is profiled and fixed offline. These recurring shapes include small projections and sequence lengths that leave efficient GPU tiles partly unused, which shape-specific kernels reclaim. The GPU and NPU share DRAM through a unified memory architecture Chen et al. (2025), so intermediate tensors pass between them on chip, and each accelerator retains most of its solo throughput while the other one runs (Section 2.2). The residual contention for memory bandwidth still shifts

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

operator speed, which separates the placement that gives every operator its fastest accelerator in isolation from the placement with the highest sustained throughput.

We present VLA-Duo, a mobile VLA execution system with three main contributions:

- A periodic GPU–NPU pipeline that splits each inference at one cut point and overlaps adjacent inferences, emitting one action chunk per period T while keeping observation-to-action latency within $2T$.
- An offline performance model and scheduler that use paired operator measurements to predict concurrent execution times, and that select the cut point, accelerator order, and execution period.
- Shape-specific GPU and NPU optimizations that increase work-group parallelism, reduce padding and memory-access overheads, and enable efficient NPU execution.

We evaluate VLA-Duo using π_0 and $\pi_{0.5}$ on Snapdragon 8 Elite and Snapdragon 8 Elite Gen 5. In short-run measurements, our system achieves 36.1–78.3% higher throughput than the ExecuTorch/QNN NPU baseline. Periodic scheduling provides 1.18–$1.25\times$ the throughput of our optimized NPU-only runtime. Output periods of 394–499 ms correspond to supported action rates of 10.0–12.7 Hz when five actions are used per chunk. The schedules selected offline remain within 0.5% of the best tested period. In separate LIBERO Liu et al. (2023) simulations with a delay from observation-to-action of ten-control-steps, training-free prefix inpainting achieves 94.5% average success with $\pi_{0.5}$, 3.0 percentage points above uncorrected asynchronous execution.

2 Background and Motivation

2.1 Vision-Language-Action Inference

Vision-language-action (VLA) policies map camera observations, language instructions, and robot states to actions Zitkovich et al. (2023); Kim et al. (2024); Physical Intelligence et al. (2025). We focus on chunked flow-matching policies such as π_0 Black et al. (2025a), which generate a sequence of future actions in each inference. One inference runs three stages in sequence: a vision encoder converts images into visual tokens, a prefix stage processes those tokens together with the instruction, and an action expert iteratively transforms noise into an action chunk. The action expert reuses the prefix attention keys and values at every sampling step.

Work on efficient VLA inference shortens this computation by changing the model. Compact backbones Wen et al. (2025); Shukor et al. (2025), token and feature reuse Xu et al. (2025b), input-dependent depth Yue et al. (2024), low-bit weights Wang et al. (2025), and parallel action decoding Kim et al. (2025) each reduce the work in one inference. VLA-Duo keeps the model unchanged and schedules the resulting computation across two accelerators, so these methods compose with it.

With K actions used from each predicted chunk and one chunk produced every T seconds, the supported action rate is K/T Hz. This throughput-derived metric is distinct from the controller's execution clock.

2.2 Heterogeneous Mobile SoCs

Mobile SoCs integrate a GPU and an NPU within a unified memory architecture. Running both accelerators concurrently raises memory-bandwidth utilization and combines their compute capacity Chen et al. (2025): while one accelerator moves data from DRAM, the other computes on data already held in its on-chip memory (Figure 1, right).

We compare each accelerator's throughput in isolated and concurrent execution using repeated $[8192, 8192] \times [8192, N]$ matrix multiplications on Snapdragon 8 Elite, with N ranging from 32 to 8192. Sweeping the output width N spans bandwidth-bound and compute-bound regimes with one operator type. Across the tested shapes, each accelerator retains most of its computation capacity during co-execution (Figure 1, left). Co-execution therefore raises aggregate throughput, and the speed of an operator depends on what the peer accelerator runs alongside it.

Mobile inference systems exploit this heterogeneity at granularities from single operators to whole models. CoDL Jia et al. (2022) splits individual operators between the CPU and GPU, llm.npu Xu et al. (2025a) offloads LLM prefill to the NPU, PowerInfer-2 Xue et al. (2024) distributes sparsely

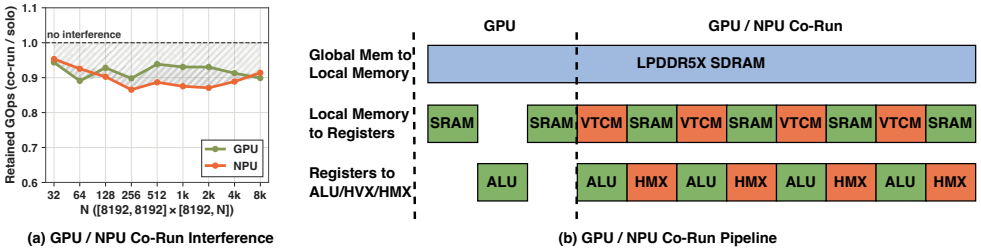

Figure 1: GPU–NPU co-execution on Snapdragon 8 Elite. Left: throughput retained by each accelerator during co-execution, relative to solo execution. Right: one accelerator moves data from DRAM while the other computes on data already in on-chip memory.

activated neuron clusters across the CPU and NPU, HeteroInfer Chen et al. (2025) partitions tensors between the GPU and NPU within one LLM inference, and Band Jeong et al. (2022) schedules independent DNNs across mobile processors. VLA-Duo places whole atoms of successive VLA inferences, which share one static graph that a single offline plan replays every period.

2.3 Asynchronous VLA Execution

A synchronous controller alternates between predicting an action chunk and executing it, so the robot waits while each prediction is computed (Figure 4). Asynchronous VLA execution overlaps action execution with inference so that the robot continues moving while the next chunk is generated. Methods such as RTC Black et al. (2025b) additionally condition the new chunk on already committed actions to maintain continuity across chunk boundaries.

Cross-inference pipelining exposes a further source of parallelism: one accelerator begins processing a newer observation while the other continues the preceding inference. This shortens the interval between completed action chunks while each inference keeps its own latency. On a mobile SoC, the achievable interval depends on how the work is divided and on how concurrent operators interfere, so the assignment that gives every operator its fastest accelerator in isolation can differ from the assignment with the highest sustained throughput. These observations motivate a periodic GPU–NPU schedule that minimizes the input period under a bound on observation-to-action latency. Conceptually, this resembles software and DNN pipeline scheduling Rau (1994); Huang et al. (2019); Narayanan et al. (2019), but here execution costs depend on the atom concurrently running on the peer accelerator.

3 System Model and Problem Formulation

We formulate mobile VLA inference as a periodic GPU–NPU co-scheduling problem under pairwise execution interference. Unlike prior heterogeneous systems that exploit parallelism within one inference Chen et al. (2025), VLA-Duo co-schedules atoms from successive inferences to minimize the steady-state initiation interval.

3.1 System Workflow

Figure 2 summarizes the system workflow. Offline, we decompose the fixed VLA graph into an atom sequence, profile execution rates in isolation and under pairwise GPU–NPU co-execution, and generate a periodic execution plan. Online, the CPU replays the plan every T, preserving the execution order within each inference while overlapping adjacent inferences across the two accelerators.

3.2 Execution Model

Given a candidate schedule, our model estimates the completion time of every atom and inference.

Atomization of VLA We represent each inference k as an ordered sequence of m scheduling atoms,

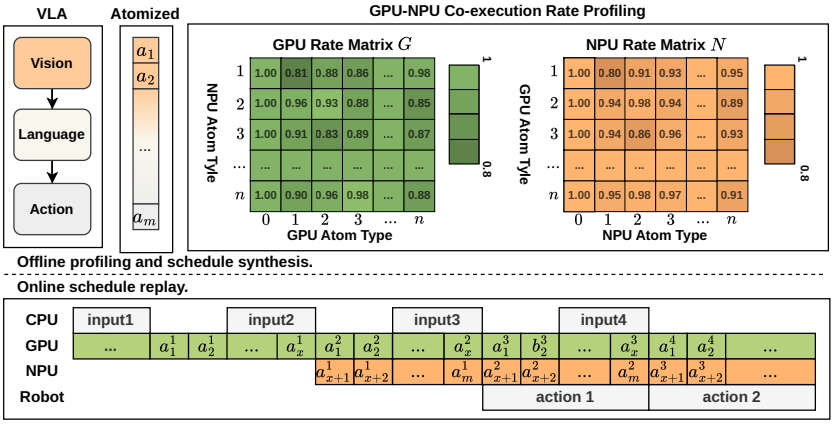

Figure 2: Overview of the System.

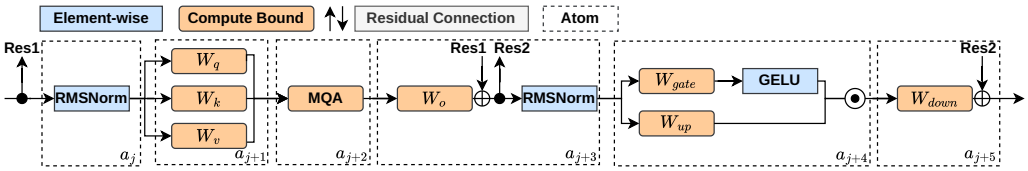

Figure 3: Atomization of a VLA Transformer block.

$$A^{(k)} = \left(a_1^{(k)}, a_2^{(k)}, \ldots, a_m^{(k)}\right). \tag{1}$$

Each atom corresponds to an operator or a fused operator group that is dispatched as one execution unit. Atom $a_{\ell+1}^{(k)}$ becomes ready only after $a_\ell^{(k)}$ completes. Because the VLA graph and tensor shapes are fixed, all inferences share the same atom sequence.

GPU–NPU co-execution. The execution rate of an atom depends on both its assigned accelerator and the atom concurrently running on the other accelerator. We capture these effects using two profiled rate matrices, G and N. Entry $G[i, j]$ denotes the rate of GPU atom type i when it co-executes with NPU atom type j, whereas $N[j, i]$ denotes the rate of the NPU atom under the same pair. Index zero represents an idle peer accelerator; therefore, $G[i, 0]$ and $N[j, 0]$ are the corresponding solo execution rates.

Quantized time. We discretize time into quanta of length τ. During each quantum, each accelerator is either idle or executes at most one atom. Let g_q and n_q denote the atom types active on the GPU and NPU during quantum q, respectively. During that quantum, the GPU and NPU complete $G[g_q, n_q]\tau$ and $N[n_q, g_q]\tau$ units of computation, respectively.

Completion time estimation. An atom may span multiple consecutive quanta. Let C_i denote the total computational work required to complete a type-i atom, with G and N expressing execution rates in work units per unit time. For an atom that starts in quantum s and completes in quantum f, its cumulative progress must satisfy the corresponding condition for GPU or NPU execution, respectively:

$$\sum_{q=s}^{f} G[g_q, n_q]\tau \geq C_i. \tag{2}$$

$$\sum_{q=s}^{f} N[n_q, g_q]\tau \geq C_i. \tag{3}$$

An atom remains on its assigned accelerator until completion.

Once the final atom $a_m^{(k)}$ completes, the action chunk produced by inference k becomes available to the robot controller for execution.

3.3 Problem Formulation

Our goal is to find the shortest period T that sustains steady-state execution, such that one new observation enters the system and one action chunk is produced in each period. Since time is quantized, we write

$$T = P\tau \tag{4}$$

where P is a positive integer. Within each period, the GPU and NPU execute atoms according to the model dependencies and the co-execution performance defined in Section 3.2.

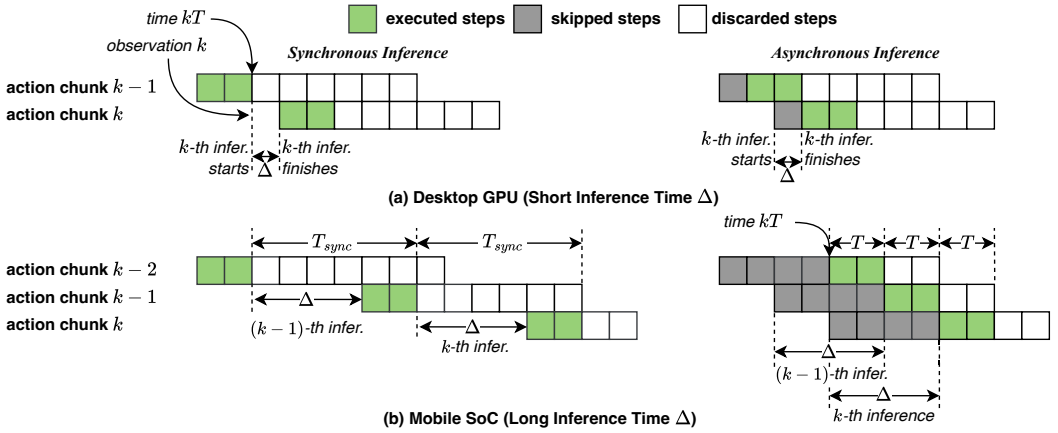

Figure 4: **Synchronous and asynchronous VLA execution under different inference latencies.** Each row represents an action chunk: green blocks denote executed actions, gray blocks denote steps skipped due to the observation-to-action delay, and white blocks denote discarded tail actions. Synchronous execution (left) has an output period T_{sync} equal to inference latency Δ plus the duration of the executed actions. Asynchronous execution (right) overlaps inference with action execution. In (b), cross-inference pipelining further overlaps successive inferences, yielding $T = \Delta/2$ in the illustrated schedule.

To exploit cross-inference parallelism across the GPU and NPU while bounding observation-to-action latency, we allow at most two in-flight inferences. As illustrated in the right panel of Figure 4(b), a new inference begins before the preceding inference finishes. Specifically, at the period boundary kT, inference k enters the system while inference $k-1$ is partway through execution. We denote the execution progress of inference $k-1$ by z, which records its position in the atom sequence and the remaining workload of its current atom.

We call a P-quantum schedule repeatable if, by the next boundary $(k+1)T$, inference $k-1$ has completed and inference k has reached the same state z. Inference $k+1$ then arrives, restoring the same execution state with the inference indices shifted by one. Repeating this overlap pattern admits one new inference and completes one action chunk every period T. Thus, the interval between successive outputs can be shorter than the latency of an individual inference, as illustrated in Figure 2(b). Each inference completes within at most two periods, giving an observation-to-action latency bound of $2T$.

We seek the smallest positive integer P for which such a repeatable schedule exists. The optimal input period is

$$T^\star = P^\star\tau, \tag{5}$$

where $P^\star$ is the minimum feasible period length. The resulting steady-state action-chunk throughput is $1/T^\star$.

4 Periodic Scheduling

As described in Section 3, each atom can run on either GPU or NPU. Finding the best schedule by trying every accelerator assignment and execution order is impractical. Besides, such fine-grained scheduling also introduces GPU-NPU handoffs between adjacent atoms, each incurring backend

synchronization and potentially requiring activation-layout conversion. We therefore use a *single-cut* schedule: each inference is divided into two contiguous segments, with one segment executed on the GPU and the other on the NPU.

4.1 Single-Cut Periodic Schedule

Consider a cut point $p \in \{0, \ldots, m\}$ that divides the atom sequence into

$$A_{\mathrm{pre}}(p) = (a_1, \ldots, a_p), \qquad A_{\mathrm{suf}}(p) = (a_{p+1}, \ldots, a_m). \tag{6}$$

For an interior cut, $p \in \{1, \ldots, m-1\}$, we consider two execution directions:

$$\begin{aligned} G \to N &: \ A_{\mathrm{pre}}(p) \text{ on GPU}, \ A_{\mathrm{suf}}(p) \text{ on NPU}, \\ N \to G &: \ A_{\mathrm{pre}}(p) \text{ on NPU}, \ A_{\mathrm{suf}}(p) \text{ on GPU}. \end{aligned} \tag{7}$$

The boundary cases $p = 0$ and $p = m$ correspond to single-accelerator execution and are retained as candidate plans.

A single-cut placement uses both accelerators while requiring only one inter-accelerator handoff per inference. In steady state, this cut also defines the progress of an inference at each period boundary. During period k, the prefix of the newly arrived inference k executes on the first accelerator, while the suffix of inference $k-1$ executes on the second accelerator. If both segments complete within the period, inference $k-1$ produces its action chunk and inference k reaches the same cut point p. The same execution pattern can therefore be repeated in the next period with the inference indices shifted by one.

4.2 Computation Period Estimation

For a cut point p and execution direction $d \in \{G \to N, N \to G\}$, we simulate the two segments concurrently. Each accelerator starts its first atom at the beginning of the period and advances to the next atom as soon as the current one completes. In each quantum, we update both atoms' remaining workloads using the pairwise rates in Section 3.2. If one segment finishes earlier, the other continues at its idle-peer rate. The simulation ends when both segments finish, yielding a computation period of $P_{\mathrm{comp}}(p, d)$ quanta.

The single handoff per inference incurs little overhead relative to the computation period. We therefore optimize the computation period and add a small allowance for the measured handoff overhead to the final execution period.

4.3 Cut Search and Plan Generation

Moving the cut changes both the accelerator workloads and the co-executing atom pairs, so balancing isolated execution times need not yield the shortest period. We evaluate all cut points in both directions using the simulation in Section 4.2, covering $2(m-1)$ heterogeneous schedules and two single-accelerator plans. Among these candidates, we select the smallest predicted computation period:

$$(\hat{p}, \hat{d}) = \arg\min_{p,d} P_{\mathrm{comp}}(p, d). \tag{8}$$

The final period $T^\star$ is obtained from $P_{\mathrm{comp}}(\hat{p}, \hat{d})\tau$ by adding the measured worst-case handoff time for the selected direction and rounding up to a multiple of τ. Single-accelerator plans require no handoff allowance.

The generated execution plan records the selected cut point, execution direction, period, and atom launch order from the simulation. The runtime repeats this plan every $T^\star$, producing one action chunk per period after warm-up.

5 Operator Optimization for Mobile Accelerators

The periodic scheduler relies on efficient atom execution on both accelerators. For a fixed task configuration, VLA inference repeats the same tensor shapes across control steps, allowing us to specialize operators and data layouts for the mobile GPU and NPU.

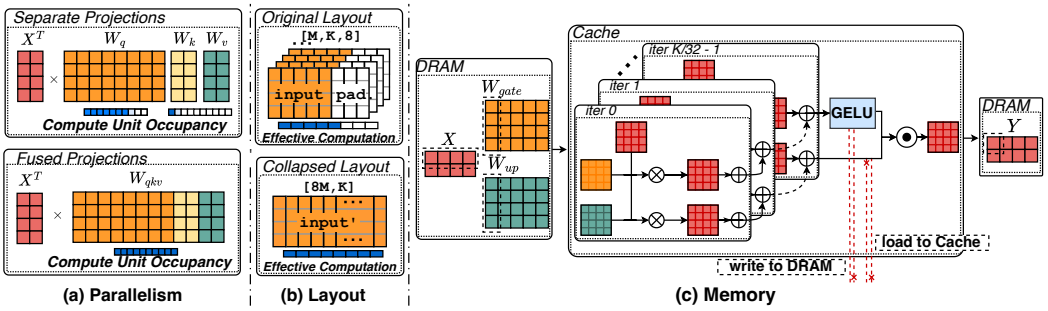

Figure 5: **Representative VLA operator optimizations.**

Parallelism. General-purpose inference frameworks can underutilize accelerators on the small operators in VLA models. For example, the 300M-parameter action expert in the π series has feed-forward and attention projections with small weight matrices, which limit available parallelism. We expose more parallel work through finer-grained tiling, GPU split-K, and operator fusion. Figure 5(a) shows how fusing three separate Q, K, V projections into one wider matrix multiplication increases the number of output tiles per launch and improves accelerator utilization.

Layout. Tensor layouts determine the matrix shapes presented to an accelerator and the kernels available for execution. We reorganize independent operations that share an operand into larger matrix multiplications to enable efficient tiled kernels. For example, the π series uses a special case of grouped-query attention in which eight query heads share one KV head. We fold the query-head dimension into the sequence dimension, converting per-head batched GEMMs into single two-dimensional GEMMs (Figure 5(b)).

Memory. After the compute optimizations above, GPU execution becomes more sensitive to memory traffic, especially when the GPU and NPU contend for shared DRAM bandwidth. We therefore fuse GELU and elementwise multiplication into the tiled gate/up projection kernel (Figure 5(c)), consuming completed tiles on chip before storing the output. Avoiding intermediate writes and rereads reduces this computation's latency by approximately 40%. We also eliminate redundant layout conversions between adjacent grouped-query attention operators that support the same layout.

6 Evaluation

6.1 Experimental Setup

Models and tasks. We evaluate the π_0 and $\pi_{0.5}$ checkpoints released by LeRobot. Manipulation is assessed on LIBERO Liu et al. (2023) across the Spatial, Object, Goal, and Long suites, each containing 10 tasks and 100 evaluation episodes drawn from ten initial states per task. Each inference takes 3 camera views at 224×224 resolution, forms 774 (π_0) or 912 ($\pi_{0.5}$) prefix tokens, runs 10 flow-matching steps, and emits a chunk of 50 actions, of which the controller executes $K = 5$.

Devices and baselines. Mobile measurements use Oneplus 13 (Snapdragon 8 Elite) and Oneplus 15 (Snapdragon 8 Elite Gen 5), with OpenCL for GPU execution and the Hexagon backend for NPU execution at W4A16 (FP16 for vision) numerical precision. We compare against llama.cpp Gerganov & llama.cpp contributors (2026), MNN Jiang et al. (2020), TVM/MLC MLC team (2023-2025), and ExecuTorch/QNN Nachin et al. (2026) at versions b8414, 3.6.0, 0.20.dev162, and 1.3.1 (QNN 2.37), respectively.

Protocol and metrics. The LIBERO controller clock is fixed at 10 Hz, so the ten-step delay of Section 6.2 corresponds to 1000 ms. The period T is the steady-state interval between consecutive action-chunk completions, and the observation-to-action latency L is the time from observation capture to release of the first action of the corresponding chunk. Unless otherwise noted, periods are averaged over three independent runs. Each run includes three warm-up inferences followed by 20 measured inferences.

Table 1: **Closed-loop success rates on LIBERO (%).** FT: fine-tuning; Δ: observation-to-action delay in control steps. †Published VLASH results use its original protocol Tang et al. (2025); its π_0 results are unavailable.

Model	Method	FT	Extra compute	Δ	Spatial	Object	Goal	Long	Average
$\pi_{0.5}$	Reference	✗	✗	0	99.0	97.0	97.0	95.0	97.0
	Naive async	✗	✗	10	91.0	89.0	95.0	91.0	91.5
	RTC	✗	✓	10	90.0	98.0	96.0	91.0	93.8
	VLASH†	✓	✗	3	94.4	98.8	93.3	91.9	94.6
	Ours	✗	✗	10	92.0	98.0	95.0	93.0	94.5
π_0	Reference	✗	✗	0	94.0	96.0	94.0	86.0	92.5
	Naive async	✗	✗	10	85.0	92.0	87.0	76.0	85.0
	RTC	✗	✓	10	83.0	98.0	90.0	78.0	87.2
	Ours	✗	✗	10	86.0	94.0	86.0	86.0	88.0

Table 2: **Mobile inference performance.** T: output period (ms); $f = 5000/T$: supported action rate (Hz, $K = 5$).

Runtime	Accelerator	Snapdragon 8 Elite				Snapdragon 8 Elite Gen 5			
		π_0		$\pi_{0.5}$		π_0		$\pi_{0.5}$	
		T	f	T	f	T	f	T	f
llama.cpp / OpenCL	GPU	5075	0.99	4854	1.03	4546	1.10	4359	1.15
MNN / OpenCL	GPU	3184	1.57	3550	1.41	4258	1.17	4754	1.05
TVM / MLC	GPU	10132	0.49	11528	0.43	8254	0.61	9154	0.55
llama.cpp / HTP	NPU	3293	1.52	3421	1.46	2442	2.05	2666	1.88
ExecuTorch / QNN	NPU	614	8.14	889	5.62	550	9.09	647	7.72
Ours, GPU only	GPU	2327	2.15	2543	1.97	2215	2.26	2432	2.06
Ours, NPU only	NPU	563	8.88	623	8.03	481	10.39	514	9.72
Ours, periodic	GPU+NPU	**450.9**	**11.09**	**498.5**	**10.03**	**394.0**	**12.69**	**436.3**	**11.46**

6.2 Manipulation Performance

Injecting a fixed delay isolates how a policy handles stale observations and joins successive chunks, separately from the timing behavior of the mobile runtime. Every asynchronous method receives the same ten-control-step observation-to-action delay, which matches the $2T$ bound of the selected plans, while the synchronous reference runs with no injected delay.

VLA-Duo conditions each chunk on the actions already committed by inpainting them into the sampling prefix, which uses no gradient guidance, no extra inference compute, and no policy training. The comparison covers uncorrected asynchronous execution and gradient-guided RTC Black et al. (2025b), which spends additional compute on each chunk. Table 1 also lists published $\pi_{0.5}$ VLASH results at $\Delta = 3$ Tang et al. (2025), measured under a different protocol and delay.

At the ten-step delay, VLA-Duo reaches 94.5% average success on $\pi_{0.5}$ and 88.0% on π_0, recovering 3.0 points over uncorrected asynchronous execution on both models while remaining 2.5 and 4.5 points below the synchronous reference. RTC reaches averages of 93.8% and 87.2% at the same delay using extra inference compute per chunk. These runs use BF16 checkpoints; Appendix B.2 reports closed-loop results at the mobile deployment precision, where $\pi_{0.5}$ retains 93.5% average success.

6.3 Mobile Inference Performance

Table 2 reports the output period T and the supported action rate $5000/T$ at $K = 5$, against llama.cpp Gerganov & llama.cpp contributors (2026), MNN Jiang et al. (2020), TVM/MLC MLC team (2023-2025), and ExecuTorch/QNN Nachin et al. (2026). Measurements follow the burst protocol of Section 6.1, taken on cooled devices before the phones reach thermal steady state. The

supported action rates are 11.09 and 10.03 Hz for π_0 and $\pi_{0.5}$ on Snapdragon 8 Elite, and 12.69 and 11.46 Hz on 8 Elite Gen 5.

Two effects produce these numbers, and the table separates them. Our single-accelerator runtimes shorten the period by 1.91–$2.18\times$ on the GPU and 5.49–$5.85\times$ on the NPU relative to llama.cpp's OpenCL and HTP backends on 8 Elite, which reflects the kernels of Section 5 together with graph and runtime optimizations. The cross-inference pipeline then adds a further 17.9–25.0% over our own NPU-only runtime.

The measured gains are 24.8–25.0% on 8 Elite and 17.9–22.2% on Gen 5. Periodic execution also trades latency for rate, since a chunk becomes available within $2T$ rather than within one period of single-accelerator execution.

6.4 Scheduling Accuracy

Table 3: **Period prediction and cut-selection regret.**

Model	T_{est} (ms)	T_{device} (ms)	Error (%)	Regret (%)
Snapdragon 8 Elite				
π_0	447	450.9	0.9	0.07
$\pi_{0.5}$	494	498.5	0.9	0.12
Snapdragon 8 Elite Gen 5				
π_0	399	394.0	1.3	0.08
$\pi_{0.5}$	429	436.3	1.7	0.44

Table 3 reports VLA-Duo's period prediction errors of 0.9–1.7% and cut-selection regret of 0.07–0.44% across the four configurations. Prediction error is normalized by the measured period.

Regret is the percentage increase in period over the best tested cut under eager admission, excluding the final drain. Appendix B.3 compares solo-profile search, VLA-Duo, and the measured oracle. Figure 6 shows layer- and step-level sweeps alongside atom-level model selections.

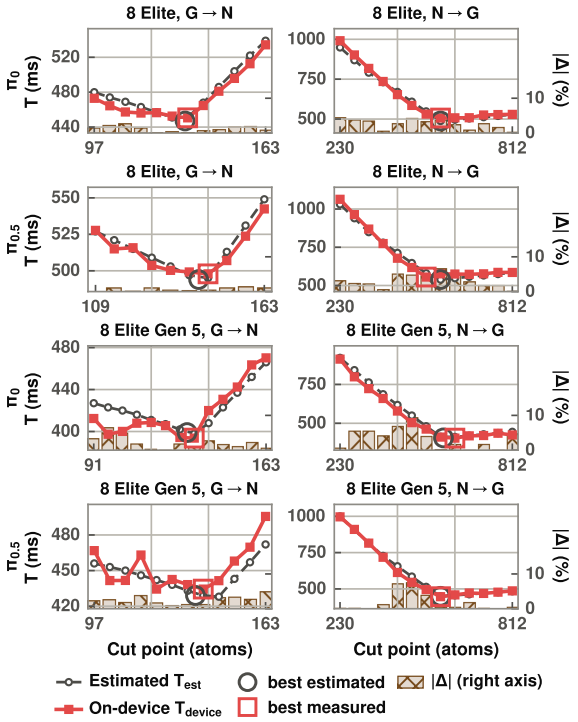

Figure 6: **Estimated and measured periods across cut points.** Outlined markers show atom-level selections (circles) and best measured cuts (squares). Bars show relative error.

7 Conclusion

We presented VLA-Duo, a periodic GPU–NPU scheduling system for mobile VLA inference that overlaps successive inferences to improve action-chunk throughput. VLA-Duo uses offline GPU–NPU co-execution profiles to select a single-cut execution plan and replays this plan at runtime while bounding observation-to-action latency within two periods. On Snapdragon 8 Elite and 8 Elite Gen 5, VLA-Duo achieves output periods of 394–499 ms and improves throughput by 17.9–25.0% over our optimized NPU-only runtime. The selected schedules incur less than 0.5% regret relative to the best measured cut. LIBERO experiments further show that prefix inpainting can mitigate the effect of delayed observations without additional training or gradient computation. These results demonstrate the potential of cross-inference heterogeneous scheduling for practical onboard VLA execution.

REFERENCES

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. π_0: A vision-language-action flow model for general robot control. In *Proceedings of Robotics: Science and Systems*, 2025a. doi: 10.15607/RSS.2025.XXI.010. URL https://www.roboticsproceedings.org/rss21/p010.html.

Kevin Black, Manuel Y. Galliker, and Sergey Levine. Real-time execution of action chunking flow policies. In *Advances in Neural Information Processing Systems*, volume 38, 2025b. doi: 10.52202/085713-1122. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/300ccb2187dedd4edcc07f7e76d8e553-Abstract-Conference.html.

Le Chen, Dahu Feng, Erhu Feng, Yingrui Wang, Rong Zhao, Yubin Xia, Pinjie Xu, and Haibo Chen. Characterizing mobile soc for accelerating heterogeneous llm inference. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, pp. 359–374, 2025.

Georgi Gerganov and llama.cpp contributors. llama.cpp: Llm inference in c/c++. https://github.com/ggml-org/llama.cpp, 2026.

Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. GPipe: Efficient training of giant neural networks using pipeline parallelism. In *Advances in Neural Information Processing Systems*, 2019.

Joo Seong Jeong, Jingyu Lee, Donghyun Kim, Changmin Jeon, Changjin Jeong, Youngki Lee, and Byung-Gon Chun. Band: Coordinated multi-DNN inference on heterogeneous mobile processors. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 235–247, 2022. doi: 10.1145/3498361.3538948.

Fucheng Jia, Deyu Zhang, Ting Cao, Shiqi Jiang, Yunxin Liu, Ju Ren, and Yaoxue Zhang. CoDL: Efficient CPU-GPU co-execution for deep learning inference on mobile devices. In *Proceedings of the 20th Annual International Conference on Mobile Systems, Applications and Services (MobiSys)*, pp. 209–221, 2022. doi: 10.1145/3498361.3538932.

Xiaotang Jiang, Huan Wang, Yiliu Chen, Ziqi Wu, Lichuan Wang, Bin Zou, Yafeng Yang, Zongyang Cui, Yu Cai, Tianhang Yu, Chengfei Lv, and Zhihua Wu. Mnn: A universal and efficient inference engine. In *MLSys*, 2020.

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.

Moo Jin Kim, Chelsea Finn, and Percy Liang. Fine-tuning vision-language-action models: Optimizing speed and success. *arXiv preprint arXiv:2502.19645*, 2025. URL https://arxiv.org/abs/2502.19645.

Bo Liu, Yifeng Zhu, Chongkai Gao, Yihao Feng, Qiang Liu, Yuke Zhu, and Peter Stone. Libero: Benchmarking knowledge transfer for lifelong robot learning. *Advances in Neural Information Processing Systems*, 36:44776–44791, 2023.

MLC team. MLC-LLM, 2023-2025. URL https://github.com/mlc-ai/mlc-llm.

Mergen Nachin, Digant Desai, Sicheng Stephen Jia, Chen Lai, Mengwei Liu, Jacob Szwejbka, Raziel Alvarez, RJ Ascani, Dave Bort, Manuel Candales, et al. ExecuTorch - a unified PyTorch solution to run AI models on-device. *arXiv preprint arXiv:2605.08195*, 2026. URL https://github.com/pytorch/executorch.

Deepak Narayanan, Aaron Harlap, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, Phillip B. Gibbons, and Matei Zaharia. PipeDream: Generalized pipeline parallelism for DNN training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 1–15, 2019. doi: 10.1145/3341301.3359646.

Physical Intelligence, Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Sergey Levine, et al. $\pi_{0.5}$: A vision-language-action model with open-world generalization. *arXiv preprint arXiv:2504.16054*, 2025. URL https://arxiv.org/abs/2504.16054.

B. Ramakrishna Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proceedings of the 27th Annual International Symposium on Microarchitecture (MICRO)*, pp. 63–74, 1994. doi: 10.1145/192724.192731.

Mustafa Shukor, Dana Aubakirova, Francesco Capuano, Pepijn Kooijmans, Steven Palma, Adil Zouitine, Michel Aractingi, Caroline Pascal, Martino Russi, Andres Marafioti, Simon Alibert, Matthieu Cord, Thomas Wolf, and Remi Cadene. SmolVLA: A vision-language-action model for affordable and efficient robotics. *arXiv preprint arXiv:2506.01844*, 2025. URL https://arxiv.org/abs/2506.01844.

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. URL https://arxiv.org/abs/2512.01031v2.

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. URL https://arxiv.org/abs/2506.07530.

Junjie Wen, Yichen Zhu, Jinming Li, Minjie Zhu, Kun Wu, Zhiyuan Xu, Ning Liu, Ran Cheng, Chaomin Shen, Yaxin Peng, Feifei Feng, and Jian Tang. TinyVLA: Towards fast, data-efficient vision-language-action models for robotic manipulation. *IEEE Robotics and Automation Letters*, 2025. URL https://arxiv.org/abs/2409.12514. arXiv:2409.12514.

Daliang Xu, Hao Zhang, Liming Yang, Ruiqi Liu, Gang Huang, Mengwei Xu, and Xuanzhe Liu. Fast on-device LLM inference with NPUs. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2025a. doi: 10.1145/3669940.3707239. arXiv:2407.05858.

Siyu Xu, Yunke Wang, Chenghao Xia, Dihao Zhu, Tao Huang, and Chang Xu. VLA-Cache: Efficient vision-language-action manipulation via adaptive token caching. In *Advances in Neural Information Processing Systems*, 2025b. URL https://arxiv.org/abs/2502.02175. arXiv:2502.02175.

Zhenliang Xue, Yixin Song, Zeyu Mi, Xinrui Zheng, Yubin Xia, and Haibo Chen. PowerInfer-2: Fast large language model inference on a smartphone. *arXiv preprint arXiv:2406.06282*, 2024. URL https://arxiv.org/abs/2406.06282.

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. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2411.02359. arXiv:2411.02359.

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. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

A AI DISCLOSURE

In this work, we used generative AI tools for finding related works and providing writing and polishing assistance. We have not used generative AI tools for ideation, drafting papers or providing mathematical claims. Generating datasets are not applicable to this work.

B ABLATION

B.1 OPTIMIZATION BREAKDOWN

Table 4 incrementally adds the three optimization stages of VLA-Duo to llama.cpp on both SoCs. The baseline, custom-kernel, and final periodic-scheduling results are repeated from Table 2; the graph-and-memory stage follows the same measurement protocol.

Table 4: **Cumulative optimization breakdown for π_0. T is the output period (ms). Speedup is relative to the baseline llama.cpp runtime on the same SoC. The final stage uses both accelerators and therefore reports one period per SoC.**

Stage	What is added	8 Elite		8 Elite Gen 5		Speed-up
		NPU	GPU	NPU	GPU	8E / Gen 5
Baseline runtime	llama.cpp (HTP / OpenCL)	3293	5075	2442	4546	1.0 / 1.0×
+ graph/memory opt.	shape alignment, copy eliminate, graph reuse	1007	4290	703	3629	3.3 / 3.5×
+ kernel opt.	our HMX/HVX and OpenCL kernels	563	2327	481	2215	5.8 / 5.1×
+ hetero.	periodic scheduling	**450.9**		**394.0**		**7.3 / 6.2**×

Graph/memory optimizations. This stage mainly optimizes data movement and graph construction. We align SigLIP tensor shapes with existing HMX kernels, eliminate redundant layout and KV-cache copies, stage FFN activations in FP16, reuse the prefix graph across inferences, and replace DSP signaling with polling. These changes reduce NPU latency by $3.3\times$ on 8 Elite and $3.5\times$ on Gen 5, with most of the gain coming from the vision encoder (2677 $\rightarrow$ 498 ms and 1921 $\rightarrow$ 305 ms). The corresponding GPU graph/layout changes provide a $1.2\times$ speedup on both SoCs.

Kernel optimizations. This stage mainly optimizes kernels for the OpenCL and Hexagon backends. On the Hexagon backend, we specialize attention, FFN, normalization, and residual kernels for the fixed VLA tensor shapes. On OpenCL, we optimize GEMM layouts and use split-K kernels to improve GPU utilization. These kernel optimizations provide an additional $1.8\times$ /$1.5\times$ speedup on the NPU and $1.8\times$ /$1.6\times$ on the GPU for 8 Elite / Gen 5, respectively.

Heterogeneous execution. The last rung runs the periodic GPU–NPU schedule of the main text; the period drops to 450.9 and 394.0 ms for 8 Elite and 8 Elite Gen 5, respectively.

B.2 CLOSED-LOOP EVALUATION AT DEPLOYMENT PRECISION

Closed-loop tests use the same precision configuration as VLA-Duo in Table 2: FP16 for vision and W4A16 for PaliGemma and the action expert. Table 5 reports both models.

At $\Delta = 10$, VLA-Duo achieves 93.5% average success on $\pi_{0.5}$, 1.0 percentage point below the BF16 result of 94.5% in Table 1. The synchronous reference decreases from 97.0% to 95.8%. These results indicate that the mobile deployment precision preserves most of the measured BF16 closed-loop success for $\pi_{0.5}$.

Table 5: **LIBERO success rates (%) for $\pi_{0.5}$ and π_0 at deployment precision.** Δ denotes observation-to-action delay in control steps; n/a: VLASH releases no π_0 checkpoint.

		$\pi_{0.5}$					π_0				
Method	Δ	Spatial	Object	Goal	Long	Average	Spatial	Object	Goal	Long	Average
Reference	0	97.0	97.0	97.0	92.0	95.8	98.0	93.0	87.0	71.0	87.2
VLASH	3	93.0	98.0	92.0	87.0	92.5	n/a	n/a	n/a	n/a	n/a
Naive	10	85.0	96.0	95.0	91.0	91.8	77.0	86.0	91.0	55.0	77.2
RTC	10	88.0	97.0	94.0	87.0	91.5	72.0	75.0	78.0	49.0	68.5
Ours	10	88.0	97.0	97.0	92.0	93.5	72.0	86.0	87.0	61.0	76.5

B.3 Effect of Co-execution Modeling on Schedule Selection

Table 6 compares the same cut search using isolated execution rates (solo-profile, without considering co-execution rates) and pairwise co-execution rates (VLA-Duo). The measured oracle is the best tested cut for each model–SoC configuration.

Table 6: **Effect of co-execution modeling on schedule selection.** p: selected cut; T_{est} and T: predicted and measured periods (ms). $\Delta = |T_{\text{est}} - T|/T \times 100\%$. Regret is measured relative to the best tested cut (oracle).

Scheduler	p	T_{est}	T	Δ (%)	Regret (%)
Snapdragon 8 Elite, π_0					
Solo-profile	135	427	455.3	6.2	1.04
VLA-Duo	132	447	450.9	0.9	0.07
Measured oracle	133	451	450.6	0.1	0.00
Snapdragon 8 Elite Gen 5, π_0					
Solo-profile	136	360	406.9	11.5	3.35
VLA-Duo	130	399	394.0	1.3	0.08
Measured oracle	133	398	393.7	1.1	0.00
Snapdragon 8 Elite, $\pi_{0.5}$					
Solo-profile	148	470	502.4	6.4	0.90
VLA-Duo	142	494	498.5	0.9	0.12
Measured oracle	145	495	497.9	0.6	0.00
Snapdragon 8 Elite Gen 5, $\pi_{0.5}$					
Solo-profile	144	386	440.3	12.3	1.36
VLA-Duo	136	429	436.3	1.7	0.44
Measured oracle	139	429	434.4	1.2	0.00

Because co-execution slows both accelerators relative to their isolated profiles, solo-profile search underestimates the measured period by 6.2–12.3%. Pairwise co-execution profiles reduce prediction error to 0.9–1.7% and cut-selection regret from 0.90–3.35% to 0.07–0.44%.