Optimizing $\pi_{0.5}$ Vision–Language–Action Robotic Model on Intel® Core™ Ultra Series 3 Processor

Asaad F Said Harshil Patel Deepak S Amrutha Dhanakumar Alex Turk Chon Ming Lee
Sergey Shumihin Anand Bodas Parual Datta Deepa Suresh Radwan Ibrahim
Vladislav Sovrasov Daniil Lyakhov Daan Krol Alfie Roddan Ashutosh Kumar
Samet Akcay Greeshma Pisharody

intel.

Abstract.

Vision–Language–Action (VLA) models unify perception, natural language understanding, and control in an end-to-end robotics framework. This enables generalization across tasks and environments with minimal task-specific customization compared with modular pipelines. Deploying VLA models at the edge is challenging as it requires high compute efficiency and substantial memory bandwidth for real-time inference. Intel® Core™ Ultra Series 3 processor, codenamed Panther Lake, addresses these requirements with a heterogeneous CPU–GPU–NPU architecture. It provides up to 180 TOPS of compute and up to 154 GB/s of memory bandwidth.

In this work, we evaluate the edge deployment of the $\pi_{0.5}$ VLA model, selected for its exceptional reliability on manipulation tasks in unfamiliar environments. We optimize $\pi_{0.5}$ for the Intel® Core™ Ultra Series 3 processor. We then benchmark its performance against leading embedded AI platforms. We show up to $2.6\times$ higher performance per watt than NVIDIA Jetson AGX Orin and up to $1.3\times$ over NVIDIA® Jetson Thor™. These results position the Intel processor as a strong platform for real-time VLA inference in edge robotics.

Demo: https://automatic-broccoli-3qk34jw.pages.github.io/
Code: https://github.com/intel-sandbox/unleashing-the-panther

1 Introduction

Robotic autonomy has traditionally relied on modular architectures that separate intelligence into distinct components such as perception, planning, and control. In this pipeline, perception converts raw sensor data into structured outputs such as object labels, pose estimates, and maps. Planning generates motion trajectories. Control then executes these trajectories with feedback control. This systematic partitioning has facilitated resilient, repeatable, deterministic, and safety-critical applications across sectors such as industrial automation, surgical robotics, and logistics, and continues to serve as a cornerstone for contemporary robotics systems Luo et al. (2024); Bai et al. (2025).

Despite their success, modular pipelines have fundamental limitations in open world and unstructured environments. Subsystems are often built or trained in isolation. This creates brittle interfaces and allows errors to compound across the perception to action loop. Adapting to new tasks or environments typically requires manual re-engineering, task specific data collection, or changes to planners and controllers. Learning based perception and dynamics models have improved robustness. However, they remain sensitive to domain shift and embodiment variation, which limits generalization across tasks and scenarios Ai et al. (2025); Hu et al. (2023).

Recent advances in Vision–Language Models (VLMs) and multimodal representation learning have accelerated the shift toward robotic foundation models. These models aim to provide reusable, general-

purpose capabilities across tasks and embodiments. Surveys on foundation models for robotics highlight two main directions: (i) using pre-trained vision and language models as high-level reasoning components, and (ii) developing robotics-specific foundation models that couple multimodal perception with action generation Jang et al. (2024); Kawaharazuka et al. (2025). Vision–Language–Action (VLA) models are a prominent example of the latter approach. VLAs integrate visual perception, natural-language understanding, and action generation into a single learned policy. This enables direct mapping from robot observations, including camera images, proprioceptive signals, and task state, together with language instructions, to generate robot actions. VLAs therefore, replace the perception–planning–control stack with a unified learning objective. They aim to capture semantic reasoning, task abstraction, and motor behavior within a shared representation.

Empirical studies show that VLA models trained on large, heterogeneous robot demonstration datasets can generalize across a wide range of manipulation tasks. They can follow open-vocabulary instructions and transfer knowledge across robot embodiments with limited fine tuning Kawaharazuka et al. (2025). Existing architectures differ in how they represent actions. Some generate autoregressive token sequences, while others use flow-based or diffusion-based continuous action generation. Despite these differences, they share a common objective. They learn end-to-end sensor-to-action policies conditioned on language Black et al. (2024); Team et al. (2025).

However, VLAs introduce challenges that are not found in traditional robotic pipelines. Large-scale data collection across tasks and platforms is costly. Inference latency can also limit real-time control. Safety and interpretability guarantees are often less explicit than in model-based systems. As a result, many practical systems adopt hybrid architectures. VLA models provide high-level reasoning or policy generation, while classical planning, control, and safety layers enforce constraints, stability, and real-time responsiveness Kawaharazuka et al. (2025); Black et al. (2024, 2025).

> **Overall.** Robotics is shifting from rigid modular pipelines toward foundation-model-driven approaches. VLA models have enabled major gains in open-vocabulary understanding, zero-shot generalization, and embodied reasoning. However, their inference latency can be a primary barrier to closed-loop, real-time deployment at the edge. This motivates optimizing VLA execution on heterogeneous compute platforms that can deliver low latency under tight power and memory constraints. In this work, we target Intel® Core™ Ultra Series 3 (codenamed Panther Lake) for real-time VLA inference. We optimize $\pi_{0.5}$ for this processor and benchmark the resulting latency and efficiency against leading embedded AI platforms.

2 Vision-Language-Action-Model: $\pi_{0.5}$

Early VLA systems integrated perception, language, and control, but showed limited real-world generalization. $\pi_{0.5}$ Black et al. (2025), developed by Physical Intelligence, addresses this gap as a foundation VLA model for open-world manipulation. It targets broad transfer across tasks, environments, and robot embodiments through diverse training data and a task architecture designed for cross-domain adaptation. $\pi_{0.5}$ can complete long-horizon household tasks, such as cleaning kitchens and bedrooms, in previously unseen environments without environment-specific retraining.

Deploying $\pi_{0.5}$ in real time imposes substantial perception throughput requirements. Modern VLA pipelines typically consume one to four camera streams at 30 frames per second (FPS). For high-precision scenarios, such as tracking moving objects, each frame must be processed to preserve task-critical accuracy. When precision requirements are lower, frames can be sub-sampled (e.g., every second or third frame) to reduce compute at the cost of temporal resolution. Figure 1 summarizes the $\pi_{0.5}$ dataflow and its three main components: Vision Encoder, Language Encoder, and Action Expert. We describe each component in the following subsections.

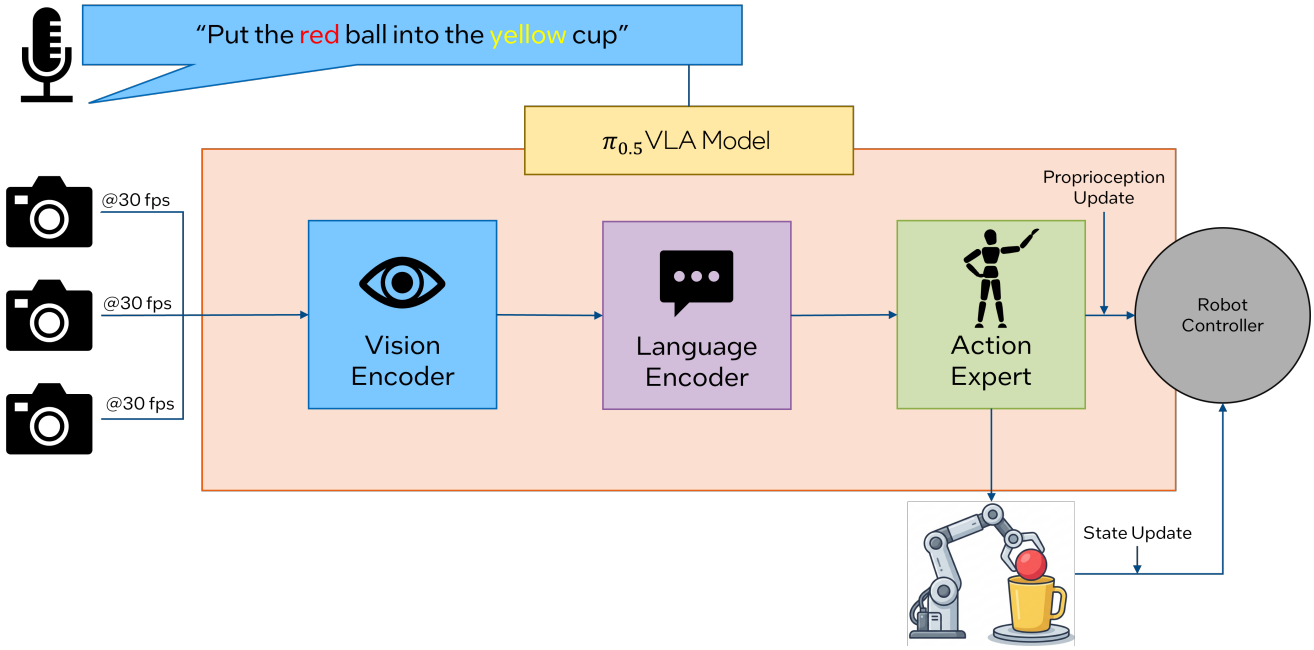

Figure 1. $\pi_{0.5}$ VLA data flow highlighting the core components—Vision Encoder, Language Encoder, and Action Expert.

2.1 Vision Encoder

The Vision Encoder (VE) Zhai et al. (2023) in $\pi_{0.5}$ processes images from one or more cameras by partitioning each frame into fixed-size patches (e.g., 14×14). Each patch is projected into a fixed-dimensional embedding, forming a sequence of visual tokens that represent objects, geometry, and appearance. A transformer backbone then aggregates these tokens to model global spatial context. The token count grows with input resolution and scales linearly with the number of camera streams. Higher-resolution inputs or multi-camera configurations therefore increase the total number of vision tokens, which increases downstream compute.

2.2 Language Encoder / VL

The Language Encoder (LE) Beyer et al. (2024) in $\pi_{0.5}$ maps natural-language instructions into semantic embeddings aligned with the vision-token latent space. These language tokens are fused with visual representations through multimodal attention, producing a shared vision–language feature space. This fusion enables grounded instruction following by linking textual directives to relevant objects and spatial context in the scene. The resulting vision–language tokens then condition downstream reasoning and guide action generation.

2.3 Action Expert

The Action Expert (AE) Black et al. (2025) is a decoder-style module designed to generate executable, multi-step action trajectories conditioned on the unified vision-language context. In practice, the AE is often a runtime bottleneck, as it repeatedly performs cross-attention over the fused vision–language tokens to produce time-coupled trajectories.

To maintain a constant computational footprint with respect to the generated action sequence, the module operates on a fixed-length set of action tokens (e.g., 50 tokens). Across transformer layers, cross-attention treats these action tokens as queries over the vision–language keys and values, selectively retrieving task-relevant spatial and semantic cues. Using an iterative flow-matching objective, the AE starts from a noisy action representation and progressively denoises it into a trajectory through alternating self-attention and cross-attention blocks. This iterative decoding aligns the action sequence with

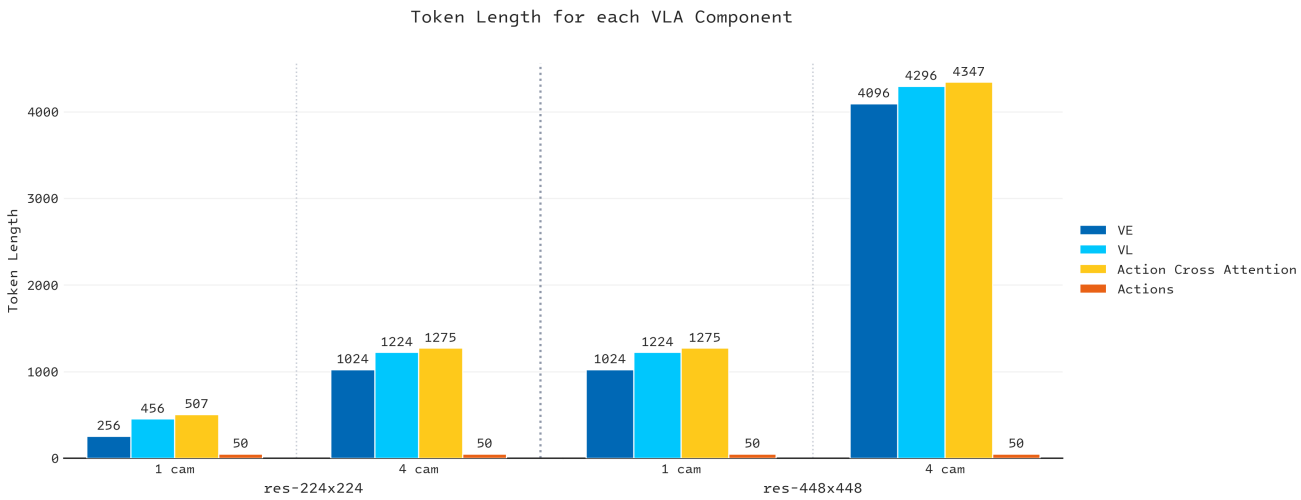

Figure 2. Token Length across several VLA components at different resolution (res) and number input camera streams (cam)

both the instruction and the observed scene, ultimately producing precise, low-latency robotic trajectories.

The token counts produced by each component of the $\pi_{0.5}$ model are analyzed as a function of image resolution, number of camera streams, and input token length, as shown in fig. 2. The resulting token scaling behavior for each stage of the pipeline is summarized below:

- Vision tokens: These scale proportionally with both resolution and the number of cameras.
- Vision–language tokens: Represent a combined encoding of visual and linguistic information.
- Action cross-attention: Computational requirements rise with interaction between fixed action tokens and the expanding pool of VL tokens.
- Action tokens: The size of each action chunk remains constant.

Although the architecture and parameter count of the model remain unchanged, computing costs and activation memory requirements are directly linked to the total token count. For this reason, input configuration, especially resolution and number of cameras, is the primary factor shaping runtime performance and overall system resource demands in VLA models.

> **Key Insight.** As input token length increases, driven by higher resolution and/or additional camera streams, the computational workload shifts toward a compute-bound scenario which illustrated in fig. 3.

This behavior is most prominent in the VE and VL modules, where large token counts amplify arithmetic intensity and utilization of compute units. In contrast, the AE component operates on a relatively small and fixed token set (50 action tokens), causing most of its layers to remain memory-bound, as data movement dominates over computation. The cross-attention layer operates over a large KV cache encompassing all vision and language tokens. However, its computational demand remains relatively low compared to the volume of data accessed. As a result, the operation is also placed in the memory-bound regime, particularly due to the small action token length.

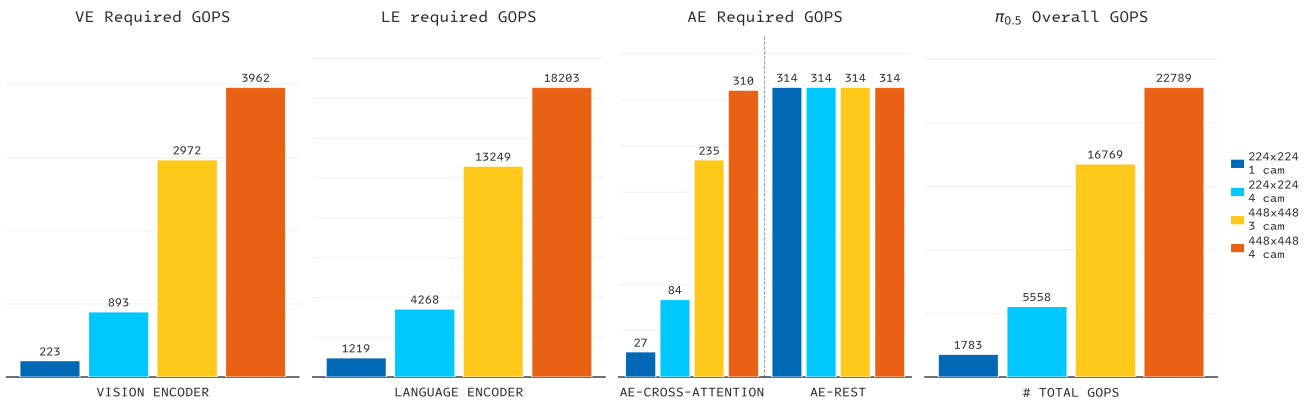

Figure 3. Absolute GOPS – various input frame resolutions and camera configurations

3 Intel® Core™ Ultra Series Series 3 processor

The Intel® Core™ Ultra Series 3 processors (codenamed Panther Lake), are Intel's first client SoCs on the Intel 18A process using a tile-based architecture to balance performance, efficiency, and scalability across mobile and embedded form factors. This processor is designed as a modular platform, integrating hybrid CPU cores, a next generation GPU with 12 Xe3 cores, and a dedicated neural processing unit (NPU) as integrated accelerators to support emerging edge AI and robotics workloads.

3.1 Hardware Configuration

The processor architecture delivers a highly balanced combination of compute capability, memory bandwidth, and energy efficiency, making it uniquely suited for real-time, multimodal robotics workloads on client edge platforms. The system in this analysis utilizes Intel® Core™ Ultra Series 3 X7 358H processor, and features a heterogeneous topology of up to 16 CPU cores—split into 4 performance cores (P-cores), 8 efficiency cores (E-cores), and 4 low-power efficiency cores (LP-E-cores) to flexibly manage latency-sensitive control tasks. Graphics and heavy compute are driven by a scalable built-in GPU Xe3 cores (PTL-iGPU) delivering up to 123 INT8-dense TOPS, while a dedicated NPU offers up to 50 INT8 TOPS optimized for power-constrained, low-latency AI inference. Backing these engines is a high-bandwidth memory subsystem supporting dual channels up to 154 GB/s, which is critical for preventing bottlenecks in data-heavy vision and perception pipelines.

This architecture enables the highly efficient execution of advanced VLA models, such as $\pi_{0.5}$ through multiple flexible deployment configurations. Developers can execute the entire VLA pipeline directly on the integrated GPU, or partition workloads across the platform's heterogeneous components to maximize system efficiency. In these hybrid configurations, the perception-heavy vision encoders and the language backbone are mapped to the GPU, the iterative action generation modules are offloaded to the NPU, and deterministic robot control tasks are handled by the CPU. This orchestrated execution model optimizes resource utilization, successfully balancing the diverse compute and strict timing requirements of next-generation robotic systems.

3.2 $\pi_{0.5}$ Performance Projection on Intel® Core™ Ultra Series 3 processor

Performance estimation for the built-in GPU is conducted using a roofline-based analytical model, consistent with recent VLA performance papers Jiang et al. (2026), which systematically examines the relationship between computational throughput and memory bandwidth. Central to this approach is the concept of Arithmetic Intensity (AIn), which is defined as the ratio of total floating-point operations (FLOPs) to bytes of memory traffic (FLOPs/byte) for each layer.

In the $\pi_{0.5}$ VLA model, each layer within the VE, LE, and AE components undergoes independent evaluation at a high level of granularity. The AIn associated with each layer determines whether its execution

on the target iGPU architecture is compute-bound (where AIn exceeds the ridge point) or memory-bandwidth-bound (where AIn falls below the ridge point).

By accounting for both compute and memory constraints, the latency of a layer can be directly estimated as the maximum of computation time and data movement time:

$$\text{Latency}_{\text{layer}} = \max\left(\frac{\text{Data Bytes}_{\text{layer}}}{BW_{\text{peak}}}, \frac{\text{Compute Ops}_{\text{layer}}}{P_{\text{peak}}}\right), \tag{1}$$

where:

- $\text{Latency}_{\text{layer}}$: Execution time of the layer (seconds)
- $\text{Data Bytes}_{\text{layer}}$: Total data transferred (Bytes)
- BW_{peak}: Peak memory bandwidth of the device (Bytes/sec)
- $\text{Compute Ops}_{\text{layer}}$: Total operations required
- P_{peak}: Peak compute throughput of the device (Ops/sec)

this formulation reflects that execution time is dominated by the slower of either memory transfer or computation.

End-to-end inference latency is calculated by aggregating the latencies of each layer, ensuring that pipeline dependencies are preserved within the VLA execution graph. To improve AIn and promote a compute-bound profile across layers, a range of optimizations are applied. These strategies are namely:

- Kernel Fusion: Combines multiple consecutive operations (e.g., MatMul, normalization, activation) into a single kernel to eliminate intermediate memory transfers, significantly reducing bandwidth usage and improving compute efficiency.
- Scaled Dot-Product Attention (SDPA) Optimization: Uses optimized SDPA optimization(e.g., FlashAttention) to compute attention in on-chip memory with tiling, minimizing DRAM access and maximizing data reuse within attention blocks.
- Memory Access and Tiling Optimization: Uses tiling and cache-aware layouts (e.g., row-major, coalesced access) to keep data on-chip, reduce DRAM traffic, and improve effective bandwidth and throughput.

These optimizations are discussed in greater detail in section 4. Together, these optimizations guide layers closer to the roofline ridge point, thereby improving throughput and decreasing latency.

Figure 4 presents PTL-iGPU performance projections utilizing two LPDDR5x channels operating at 9.6 GT/s (154GB/s) across multiple camera configurations. The VLA model is disaggregated into its principal component: VE, LE, and AE to emphasize compute and memory bottlenecks. Component analysis by AIn reveals positions relative to the roofline ridge point across different resolutions and camera scales. Projections also assess how multi-precision settings (W16A16, W8A16, W8A8) affect FLOP counts and memory traffic; using lower precision typically increases arithmetic intensity and shifts layers from bandwidth- to compute-bound. This helps identify key bottlenecks, sensitivity to precision, and opportunities to improve utilization via dataflow or quantization.

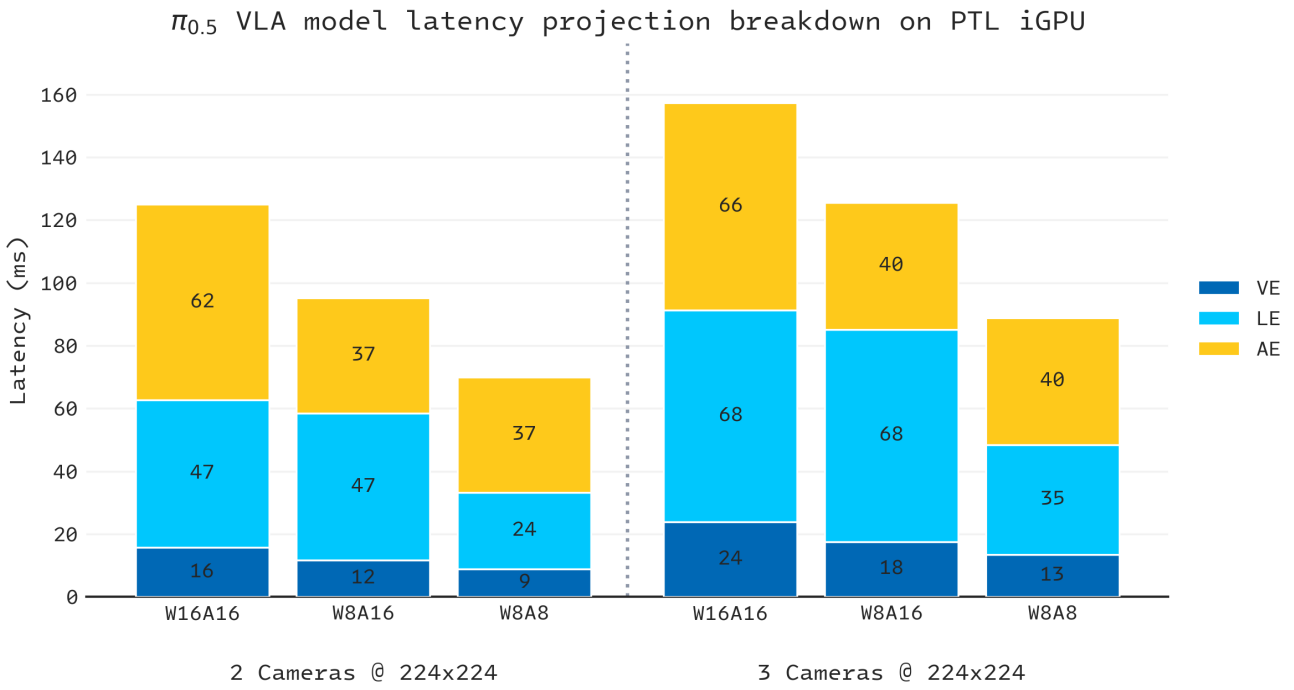

Figure 4. Latency projection on VLA components using PTL-iGPU.

Key Insight. The heterogeneous compute architecture of Intel® Core™ Ultra Series 3 - P-cores, E-cores, an Xe3 iGPU, and a dedicated NPU - enables flexible partitioning of VLA pipeline stages to best match each component's compute and timing characteristics. Combined with optimizations such as kernel fusion, SDPA tiling, and quantization, these strategies raise arithmetic intensity across layers, shifting execution from memory-bandwidth-bound toward compute-bound operation and maximizing utilization of available peak throughput.

4 $\pi_{0.5}$ Performance Measurement on Intel® Core™ Ultra Series 3 processor

Efficient deployment of the $\pi_{0.5}$ VLA model requires careful optimization across model representation, execution graph, and hardware utilization. In this section, a series of systematic optimizations are applied to improve inference performance on Intel platforms while preserving model accuracy. These optimizations span model enabling in Pytorch, conversion to OpenVINO intermediate representation (IR), precision tuning, architectural restructuring for multi-camera inputs, and runtime execution improvements. The following sections describe each optimization stage and its impact on latency, throughput, and numerical fidelity.

4.1 PyTorch Performance

The source implementation of $\pi_{0.5}$ in PyTorch Paszke et al. (2019) can be run on the target hardware using an Intel-optimized version of the framework Intel Pytorch. Initial measurements demonstrate latency of 555 ms per chunk on the hardware and model architecture configuration introduced earlier in the paper (3 cameras, 64 input tokens, and action chunk size 50, FP16 data type). One further optimization, that can be done in PyTorch, is applying `torch.compile()` to the model to reduce overhead of eager execution model. For now, `torch.compile()` is out of scope of this work, as we concentrate on inference performance in edge deployment environments, not training.

4.2 OpenVINO Model Quantization

The model optimization workflow begins with conversion of a trained PyTorch $\pi_{0.5}$ VLA model Black et al. (2025) to the OpenVINO IR format. This is performed using the OpenVINO Model Optimizer, which converts the model via ONNX into IR OepnVINO Developer Guide. During this stage, model weights are converted from FP32 to FP16 precision, reducing the memory footprint by approximately 50% while preserving inference accuracy. The resulting FP16 model serves as a consistent baseline for subsequent optimization and benchmarking. Building on FP16, several quantization techniques were employed to enhance performance. INT8 weight-only compression reduced the model size by half and improved inference speed by more than 30%, all without needing calibration data. For clear reference, methods and setups are described in Deploying $\pi_{0.5}$-DROID on Intel XPU with PyTorch.

4.3 Accelerating Vision Encoder

In the $\pi_{0.5}$ baseline architecture, each camera stream is processed independently using identical SigLIP vision encoders, resulting in redundant computation and repeated parameter fetches from main memory. To address this inefficiency, the architectural topology is refactored to optimize multi-camera ingestion within the OpenVINO execution runtime. Since the SigLIP weights are shared across all camera streams, inputs from multiple cameras are batched into a single unified tensor and executed in a single SigLIP inference call. This restructuring eliminates duplicate weight transfers over the memory bus and reduces parameter-bound memory bandwidth overhead. By exploiting batch-level concurrency, the approach lowers compute redundancy and improves overall hardware utilization on AI inference engines. In dual-camera configurations, this reduces redundant parameter traffic by approximately 50%, while in triple-camera setups the reduction approaches 66.7%, resulting in more efficient execution without modifying model semantics.

4.4 Memory Layout Optimization for General Matrix Multiplication

General Matrix Multiplication (GEMM) operations constitute the primary computational bottleneck of the $\pi_{0.5}$ VLA model architecture. Quantitative profiling OepnVINO Developer Guide reveals that GEMM-based layers account for more than 99% of the total compute operations in VLA model, with the Vision-Language Model (VLM) backbone and the Action Expert module consuming 95% and 4% of the compute budget, respectively. Non-GEMM operations, including normalization and element-wise activations, represent the remaining 1%. Given this massive computational density, optimizing memory access efficiency is critical to achieving high hardware utilization.

For large weight matrices that exceed the effective capacity of the on-chip GPU cache, a row-major memory layout is utilized to accelerate execution. This layout preserves DRAM page locality by ensuring contiguous data storage along the row dimension, which allows the hardware memory controller to issue highly sequential memory transactions during block-tiled matrix multiplication. Unlike transposed layouts that necessitate strided access across the column dimension, the row-major arrangement minimizes frequent DRAM row activation and precharge cycles. By reducing memory controller overhead and preventing page thrashing, which otherwise degrades effective bandwidth during large matrix processing, this optimized layout systematically improves sustained memory throughput and enhances overall GEMM performance within memory-bound regimes.

4.5 Scaled Dot-Product Attention Kernel Fusion Optimization

Optimizing the execution of scaled dot-product attention (SDPA) is a key focus in the deployment pipeline. The SDPA kernel fusion via OpenVINO GPU plugin fuses redundant broadcast and reshape operations that precede SDPA into a single, streamlined kernel. By eliminating unnecessary intermediate memory allocations and metadata handling, this approach significantly reduces superfluous data movement. By consolidating multiple operations into a unified SDPA kernel, this optimization significantly lowers memory bandwidth consumption and mitigates cache pressure by avoiding explicit ma-

terialization of broadcasted tensors. Furthermore, kernel dispatch overhead is reduced by collapsing multiple execution nodes into a single operation, improving overall GPU efficiency. Additional latency reductions are achieved by performing output transpositions directly within register space, thereby eliminating costly intermediate memory passes.

4.6 Gated Multi-Layer Perceptron Fusion

To optimize the execution of Gated Multi-Layer Perceptron (Gated-MLP) blocks within the $\pi_{0.5}$ model, the OpenVINO GPU plugin implements a specialized FuseGatedMLP graph transformation pass. This optimization pass detects the characteristic three-matrix-multiplication subgraph, comprising the: gate projection, a Swish activation function, an element-wise multiplication, and the subsequent up- and down-projection layers. By consolidating this multi-stage topology into a single, unified GatedMLP primitive executed via an optimized oneDNN kernel, the transformation entirely eliminates multiple sequential GPU kernel launches. Collapsing these independent execution nodes into a single dispatch completely removes intermediate global memory allocations for transient tensors and significantly mitigates GPU command submission overhead.

The latency reduction yielded by this graph fusion is particularly pronounced within the iterative $\pi_{0.5}$ action head, which executes a dense sequence of small-dimension Gated-MLP operations. In the baseline, non-fused configuration, each discrete kernel invocation incurs a non-negligible dispatch latency of approximately 104 us. Across repeated sequence evaluations, this overhead accumulates rapidly and severely restricts hardware throughput. Consolidating these operators not only lowers cumulative dispatch latency but also transitions the workload from being memory-bandwidth or latency-bound to compute-bound. Consequently, the FuseGatedMLP transformation notably reduces end-to-end latency by minimizing launch overhead and enhancing data locality, all while preserving the model's exact computational integrity.

4.7 Layer-wise Latency Profiling

Inference performance is analyzed using the OpenVINO PERF_COUNT API, enabling per-layer latency breakdown and kernel-level attribution across $\pi_{0.5}$ model components. On the Xe3p iGPU, latency is dominated by VLM attention and feed-forward layers, which are compute-bound and rely on FP16 GEMM kernels. The Gated-MLP in AE shows memory-bandwidth-limited behaviour with high utilization due to repeated denoising steps, while the SigLIP vision encoder is also compute-bound. In contrast, AE attention projection layers are affected by dispatch overhead due to small token sizes, leading to lower bandwidth efficiency.

4.8 Accuracy Evaluation Methodology

To assess the impact of these optimizations on model accuracy, a parallel validation framework computes the Mean Squared Error (MSE), Mean Absolute Error (MAE), and Cosine similarity between the trajectory outputs of the baseline PyTorch model and the optimized OpenVINO IR variants. By passing identical validation sequences through both configurations, the framework systematically quantifies real-valued prediction deviation. This comparison provides a quantitative measure of output deviation and enables rapid validation, particularly for workloads where full simulation or real-world validation is not feasible—such as Droid robotic manipulation tasks where deterministic closed-loop simulation environments are unavailable prior to physical deployment. In addition, we leverage the OpenPI Libero simulator Physical Intelligence LIBERO to evaluate task-level performance of the Pi0.5-Libero finetuned model. The simulator provides a standardized benchmark suite (e.g., Libero Spatial, Object, and Goal tasks) to assess policy generalization and robustness across diverse manipulation scenarios. The validation pipeline keeps the original preprocessing and postprocessing from the policy stack, while replacing the policy forward pass with OpenVINO inference and serving actions through the same interface expected by the simulator. It enables repeatable evaluation under controlled conditions, including multi-task rollouts and success-rate aggregation across episodes, thereby complementing metric-based vali-

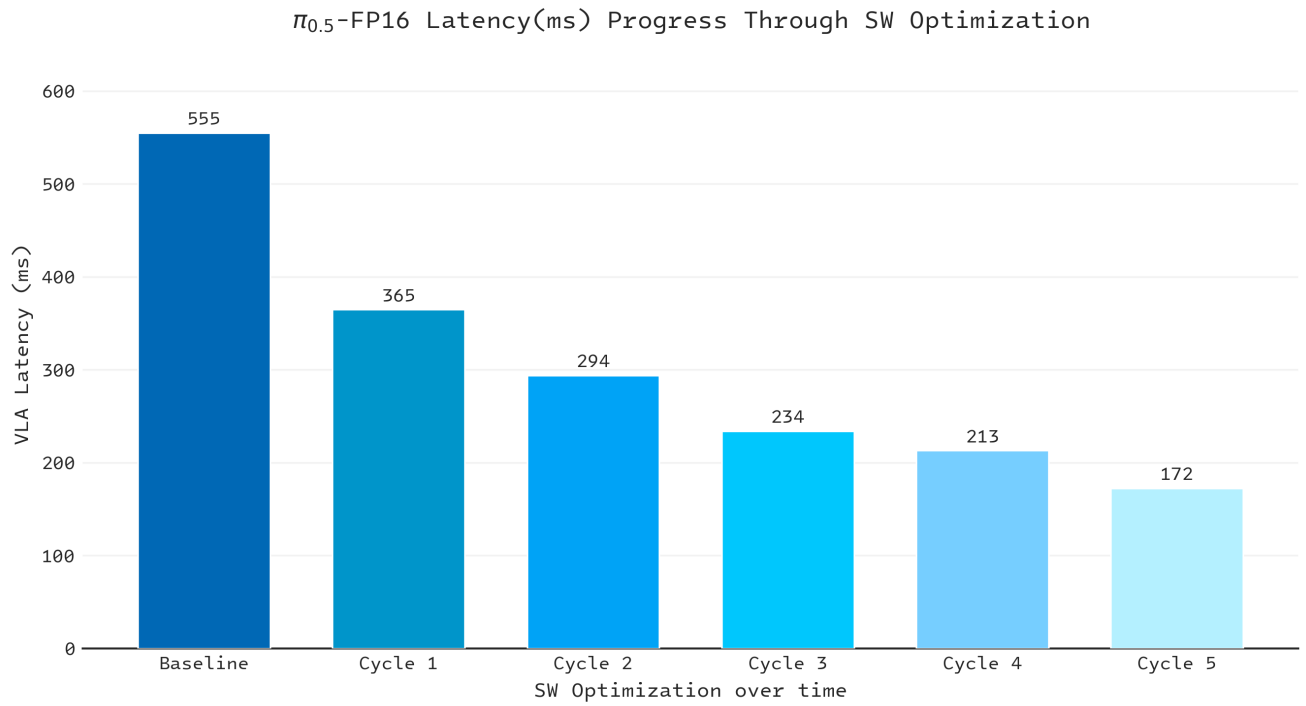

Figure 5. $\pi_{0.5}$ VLA Model Optimization Progress on PTL-iGPU. Configurations: 3 cameras at 224x224, 64 input token length, 10 denoising and FP16 precision

dation with system-level insights. More details are provided in Configuration Guide for the Optimized OpenVINO Model.

Figure 5 presents the optimization progress of the $\pi_{0.5}$-Droid model configured with three cameras at a resolution of 224x224, an input token length of 64, 10 denoising iterations, and FP16 precision. The initial latency recorded was 555 ms, which was subsequently reduced to 172 ms—aligning with our earlier projections. This improvement represents a 3.2x increase in processing speed following the implementation of the aforementioned optimization strategies. Using the simulator setup Physical Intelligence LIBERO, the optimized $\pi_{0.5}$ OpenVINO model (FP16) achieves an average success rate of 90% across evaluated tasks on PTL system.

> **Key Insight.** Through a layered optimization pipeline—spanning OpenVINO model conversion, INT8 quantization, batched vision encoding, memory layout tuning, and kernel fusion—end-to-end inference latency is reduced from 555 ms to 172 ms, a 3.2× speedup, while preserving a 90% task success rate on the Libero benchmark. This demonstrates that hardware-aware software optimizations, rather than architectural changes, are the primary lever for closing the gap between theoretical peak performance and real-world VLA deployment on client edge platforms.

5 Heterogeneous Execution for $\pi_{0.5}$ Inference

In the baseline approach, the VE, LE, and AE components run one after another on PTL-iGPU, causing high frame latency due to their cumulative processing time. The $\pi_{0.5}$ model generates of 50 action chunks with each inference, and the policy executes a set of actions before another inference is triggered (refill). Every refill causes the robot to pause for the total wall time required by all three stages. No inference occurs between refills, so there is no opportunity for in-frame contention—any overlap must happen across refills instead.

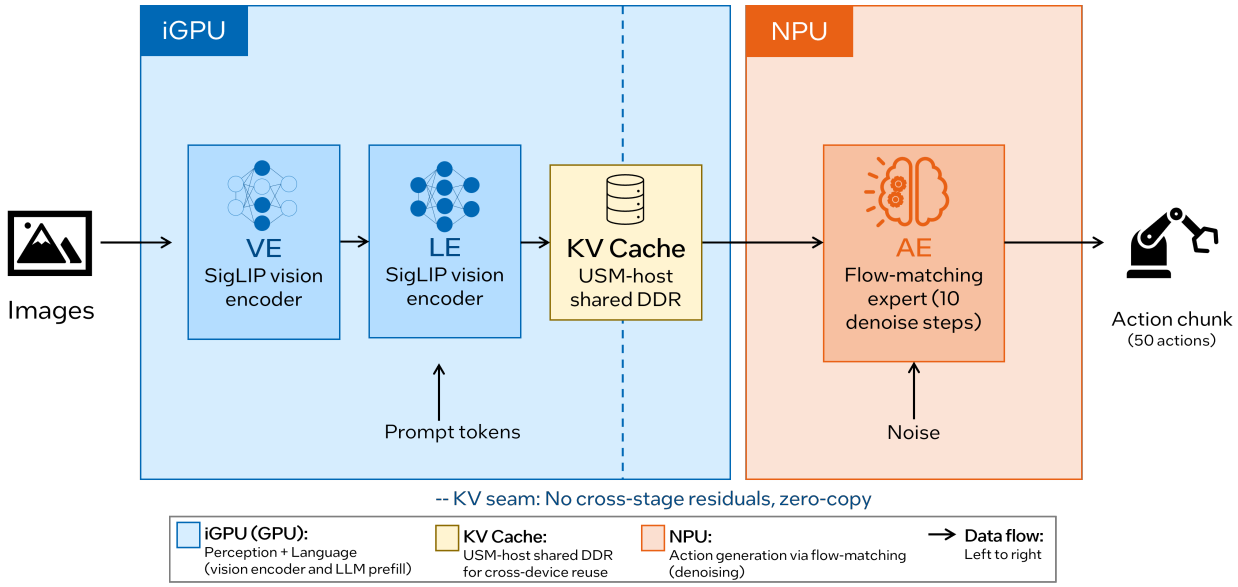

Figure 6. $\pi_{0.5}$ decomposition for heterogeneous execution: VE and LE on the iGPU, AE on the NPU, with the per-layer KV cache as the only cross-device handoff in shared system memory.

$\pi_{0.5}$ VE and LE are compute-bound, GEMM-heavy tasks that fit well on the iGPU, similar to LLM prefill. AE, on the other hand, involves smaller MatMul operations repeated frequently (10 denoising steps), with a moderate bandwidth-to-compute ratio—making it better suited for NPU offloading like an LLM decode step. While splitting the pipeline adds minor overhead, running AE on the NPU saves power and frees up the iGPU for other tasks.

Figure 6 illustrates the system's decomposition: VE and LE operate on the iGPU, while AE is deployed on the NPU. The KV cache resides in shared memory, serving as the sole point of cross-device data transfer. This configuration leverages a well-defined architectural boundary between the LE and the AE; specifically, LE generates a per-layer KV cache that AE utilizes along with its own noise input. This approach is applicable to any VLA featuring a separable action head conditioned on backbone KV (such as the pi-family) but is not suitable for autoregressive action-token models like the RT-2 family. Notably, no adjustments to model weights or retraining are necessary. Such heterogeneous execution is feasible in PTL system since all IPs including NPU and iGPU share physical memory, which allows zero-copy KV handoff without consuming an additional fraction of system memory bandwidth or adding host-CPU pressure. The OpenVINO USM-host (Unified Shared Memory) remote-tensor API can be used to expose a single physical memory buffer to both plugins: each KV tensor is allocated once via the GPU plugin context and bound as LM's output on the iGPU and AE's input on the NPU. Neither plugin issues a copy - both devices read and write the same pages.

5.1 Zero-Copy Between NPU and GPU

Heterogeneous Computing with Zero-Copy Between NPU and GPU provides a detailed explanation of the implementation process for zero-copy handoff between the GPU and NPU in the $\pi_{0.5}$ model. $\pi_{0.5}$ employs a real-time chunking (RTC) algorithm to address refill stalls by framing asynchronous chunk generation as an inpainting task. As the robot executes its current set of actions, RTC concurrently generates the subsequent chunk across multiple controller steps. Once the new chunk is prepared, the initial slots are aligned with the tail of the previous chunk that the robot was executing during the inference window, and RTC freezes them to those previous-chunk actions and inpaints the remainder of the new chunk. Partial attention is applied over a brief overlap to facilitate seamless transitions. While RTC demonstrates robust performance even in the presence of significant inference delays, its accuracy diminishes when the inference delay grows large relative to the chunk size, since the inpainting window

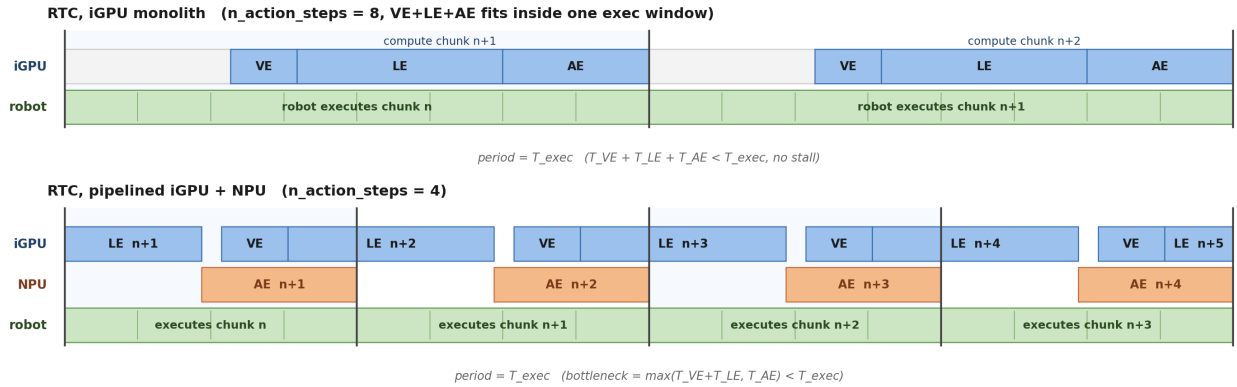

Figure 7. RTC steady state per exec window. Top: monolithic iGPU, $VE \rightarrow LE \rightarrow AE$ inside one window. Bottom: split $iGPU + NPU$, AE on the NPU overlaps $VE + LE$ on the iGPU.

shrinks.

Under synchronous chunking, the primary constraint is the refill speed. In a real-time chunking (RTC) setting, robot operation continues uninterrupted as long as each inference fits within the execution window, defined as $T_{\text{exec}} = n_{\text{action steps}} \times T_{\text{step}}$. When $T_{\text{VE}} + T_{\text{LE}} + T_{\text{AE}} \leq T_{\text{exec}}$, a monolithic iGPU solution suffices, as depicted in fig. 7, panel 1, where the entire $VE \rightarrow LE \rightarrow AE$ process chain is contained within a single window. In this scenario, partitioning does not reduce latency; only architectural advantages are achieved.

However, the execution budget may decrease below the cumulative duration of the serial stages in two scenarios: either by selecting a smaller $n_{\text{action steps}}$ to enhance reactivity in tasks requiring frequent contact, or due to increased model complexity or control rates, resulting in larger T_{VE} and T_{LE} values. Should the budget fall below $T_{\text{VE}} + T_{\text{LE}} + T_{\text{AE}}$, the monolithic implementation can no longer fit within one execution window, causing the robot to stall at each refill - precisely the situation RTC is designed to prevent.

Deploying AE on the NPU concurrently with VE+LE on the iGPU reduces the per-refill bottleneck from $T_{\text{VE}} + T_{\text{LE}} + T_{\text{AE}}$ to $\max(T_{\text{VE}} + T_{\text{LE}},\ T_{\text{AE}})$. This approach enables configurations of $n_{\text{action steps}}$ where the monolithic iGPU may stall, but the split between devices maintains consistent performance. As illustrated in Figure 7, panel 2, during steady state operation, the NPU denoises chunk $n{+}1$ while the iGPU is processing VE and LE for chunk $n{+}2$; the KV cache transitions at the LE$\rightarrow$AE seam via shared memory without requiring data copying. The observable period is T_{exec}, rather than the sum $T_{\text{VE}} + T_{\text{LE}} + T_{\text{AE}}$. However, simultaneous operation of both devices is not without cost: since the iGPU and NPU share a single memory controller, each device experiences reduced throughput under concurrent workloads compared to isolated measurements. Although this slowdown is significantly less than the performance gains achieved through overlap, it should be noted that $\max()$ represents an upper bound on savings, not an exact forecast.

On LIBERO-10 (200 episodes across all 10 tasks), the GPU+NPU serial split matches the monolithic iGPU baseline in success rate while freeing the iGPU. Overlapping the VLM with one environment step further pushes per-chunk latency below the monolithic baseline, at a small cost in success on the longest-horizon tasks.

Key Insight. Offloading the Action Expert to the NPU while the iGPU concurrently processes the Vision and Language Encoders transforms the per-refill bottleneck from a serial sum $T_{\text{VE}}+T_{\text{LE}}+T_{\text{AE}}$ into a parallel maximum $\max(T_{\text{VE}}+T_{\text{LE}},\ T_{\text{AE}})$. This heterogeneous split, bridged by zero-copy KV-cache handoff over shared system memory, enables the robot to sustain uninterrupted execution under RTC in configurations where a monolithic iGPU deployment would stall, without sacrificing task success rate.

6 Benchmarking and Competitive Analysis

This section presents an empirical benchmarking and competitive analysis of the $\pi_{0.5}$ model across three edge computing platforms: the Intel® Core™ Ultra Series 3 processor, the NVIDIA Jetson AGX Orin 64GB, and the next-generation NVIDIA® Jetson Thor™ (T5000). Hardware specifications for each architecture are detailed in table 1

Table 1. Hardware Configurations of Target Systems

Platform	NVIDIA Jetson AGX Orin 64GB	NVIDIA® Jetson Thor™ T5000	Intel® Core™ Ultra Series 3 X7 358H (Panther Lake)
TDP (W)	15–60	40–130	15–65
CPU	12-core Arm® Cortex®-A78AE	14-core Arm® Neoverse-V3AE	16-core (4P + 8E + 4LP-E)
Memory (GB)	64 GB 256-bit LPDDR5	128GB 256-bit LPDDR5	32GB (2x16GB LPDDR5 8533 MT/s)
Storage (GB)	64GB eMMC 5.1	1 TB NVMe SSD	1 TB NVMe SSD
Operating System	Ubuntu 22.04.5 LTS	Ubuntu 24.04.3 LTS	Ubuntu 24.04.4 LTS
Kernel version	5.15.148-tegra	6.8.12-tegra	6.17.0-14-generic
Jetpack version	6.2.1	7.0	N/A
CUDA version	12.6	13.0	N/A
OpenCL Compute Runtime Version	N/A	N/A	26.01.36711.4
System Peak AI Compute (INT8)	275 INT8 (Sparse)	1035 INT8 (Sparse)	180 INT8 (Dense)
GPU pTOP (INT8-Dense)	85	517	122
Peak BW (GB/s)	204	273	153
Capacity (GB)	64	128	128

The evaluation provides a comprehensive summary of benchmarking results and total cost of ownership (TCO) analysis, incorporating key metrics such as performance and performance per watt across different power budgets. These indicators deliver valuable insights into the AI capabilities of each system assessed, providing a more comprehensive understanding of system efficiency from the perspective of performance, cost, and energy consumption.

The $\pi_{0.5}$ model employs the droid variant, configured with three input cameras at a resolution of 224x224 pixels, a language sequence length of 64 tokens, and applies 10 denoising steps with BF16/FP16 precision. System efficiency is evaluated under two power settings: Max TDP, which imposes no power limitation, and ISO-TDP, where system power is capped at 40W. The subsequent subsections will present performance analyses for both the standard PyTorch implementation and the optimized variant on Intel and NVIDIA platforms.

6.1 $\pi_{0.5}$ Model Performance: PyTorch XPU vs CUDA

This section focuses on benchmarking of the $\pi_{0.5}$ DROID model on both NVIDIA and Intel platforms using PyTorch 2.12. Figure 8 presents the latency results for $\pi_{0.5}$ at max TDP and 40W settings across all platforms used in this study. The Jetson AGX Orin achieved an inference time of 1034 ms at maximum power configuration, which increased to 1530 ms at 40W configuration. The Jetson Thor platform delivered 287 ms at maximum power and 408 ms at 40W. On the PTL-iGPU on Intel platform with PyTorch XPU at 40W, inference latency was recorded at 294 ms, within 2.4% of Jetson Thor's full-power performance, indicating that Intel XPU provides comparable results with reduced platform cost and power requirements. Furthermore, the Intel processor at 40W outperformed the Jetson AGX Orin at both 60W and 40W by factors of 3.5× and 5.2×, respectively, underscoring the efficacy of XPU as a robust solution

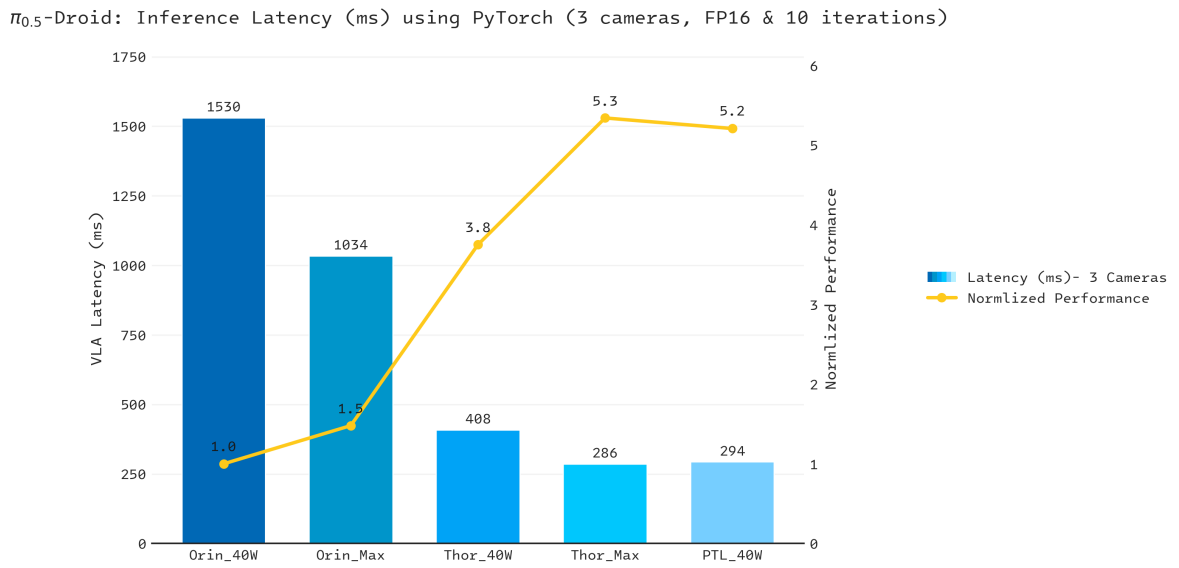

Figure 8. $\pi_{0.5}$ model latency using PyTorch on PTL-iGPU, Jetson AGX Orin, and Jetson Thor

for edge deployment of diffusion-based robotic policies.

6.2 Optimized $\pi_{0.5}$ Model Performance: OpenVINO vs TensorRT

6.2.1 Performance at Maximum TDP

The initial configuration leverages the maximum power limits of each system: 65W for Intel platform, 60W for Jetson AGX Orin, and 130W for Jetson Thor. Figure 9 presents benchmarking data, including $\pi_{0.5}$ VLA latency in milliseconds, and normalized Performance per watt relative to Jetson AGX Orin. The Intel platform based system demonstrates superior latency performance compared to Jetson AGX Orin, achieving up to 1.5 times faster results, while being slightly behind Jetson Thor at 0.9 times its performance. Notably, the Intel platform surpasses both Jetson AGX Orin and Jetson Thor with up to 1.3 times greater Performance per watt.

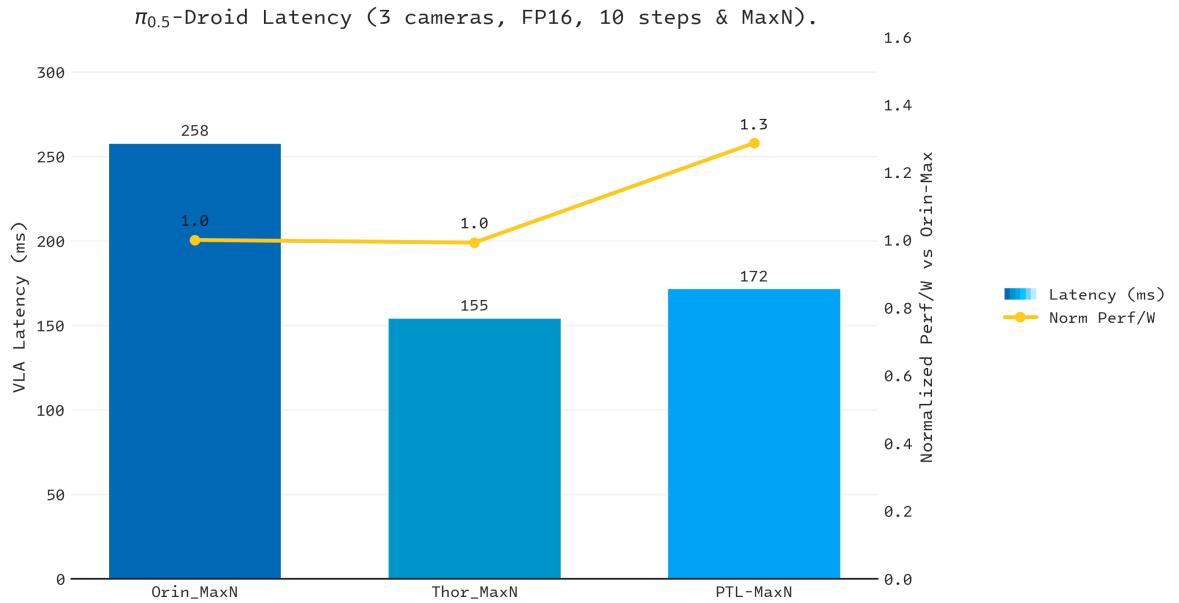

Figure 9. Competitive Analysis of $\pi_{0.5}$ model on Intel platform, Jetson AGX Orin, and Jetson Thor at Max TDP

6.2.2 Performance at ISO-TDP (40W)

In the ISO-TDP configuration, each system's SoC power is limited to a maximum of 40W to evaluate performance under a constrained power budget. Under these conditions, the Intel processor platform demonstrates exceptional results as shown in Figure 10, achieving up to 2.6 times better latency than Jetson AGX Orin and 1.1 times better than Jetson Thor. Furthermore, it outperforms both NVIDIA systems in efficiency, delivering up to 2.6 times higher Performance per watt compared to Jetson AGX Orin and Jetson Thor. The Intel platform demonstrates superior latency, performance per watt.

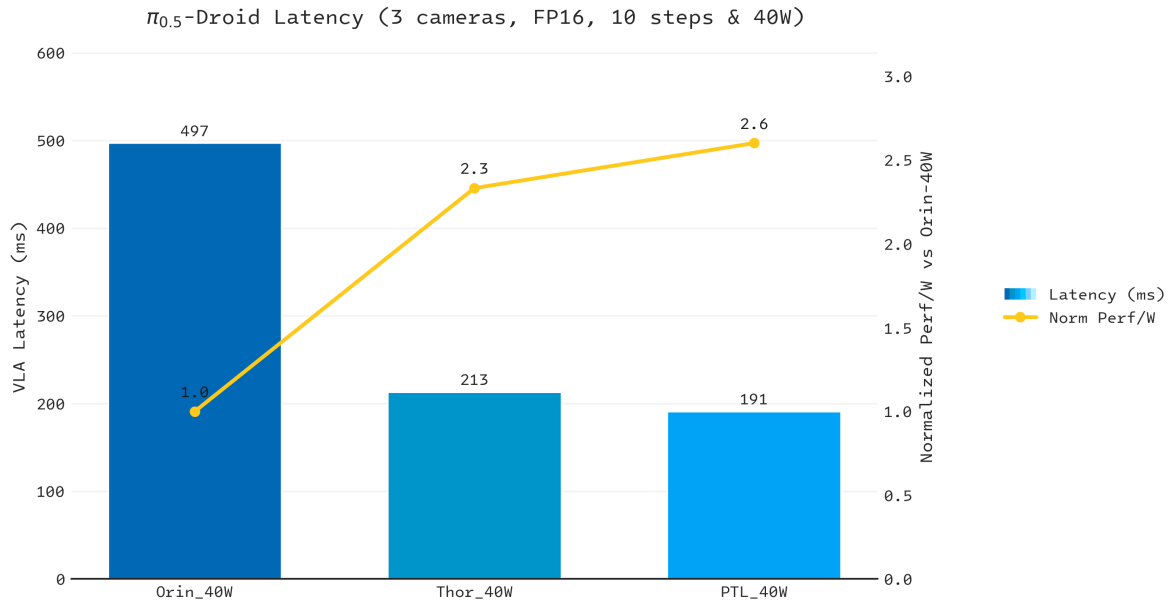

Figure 10. Competitive Analysis of $\pi_{0.5}$ model on Intel platform, Jetson AGX Orin, and Jetson Thor at 40W TDP

7 Conclusion

This study establishes that effective deployment of $\pi_{0.5}$ VLA models on edge platforms necessitates a holistic approach encompassing model architecture, dataflow optimization, and hardware-aware execution. Through comprehensive analysis of the computational and memory demands of VLA components, we have enabled targeted optimizations that enhance inference efficiency without compromising model accuracy. The implementation of a heterogeneous execution strategy on Intel® Core™ Ultra Series 3 processor, where vision and language processing are allocated to the GPU and iterative action generation is offloaded to the NPU, demonstrates significant performance improvements. The integration of zero-copy KV cache sharing facilitates parallel processing and minimizes data movement, yielding reduced end-to-end latency.

Empirical benchmarking shows that optimizing the $\pi_{0.5}$ VLA model on the PTL platform delivers better latency, Performance, performance-per-watt, and overall cost efficiency than competitors' systems. Notably, the Intel PTL platform outperforms NVIDIA Jetson AGX Orin and Jetson Thor across several key metrics. At maximum TDP, PTL delivers up to 1.5× lower latency than Jetson AGX Orin and approaches Jetson Thor performance, while providing up to 1.3× higher performance-per-watt. These findings underscore the critical role of heterogeneous, memory-efficient execution in scaling VLA models for real-time robotic applications at the edge.

Future research will be directed toward generalizing this framework to a wider spectrum of VLA architectures, investigating advanced GPU–NPU co-optimization methods, and enhancing scalability for higher-resolution and multi-camera scenarios. Additional work will focus on the development of improved low-precision quantization techniques, refined runtime scheduling for dynamic workloads, and

comprehensive system-level optimizations. These efforts aim to further decrease latency and bolster real-time responsiveness, thereby advancing the deployment of VLA models in increasingly complex and demanding robotic environments.

Acknowledgements

The authors would like to thank Arend Jan Kramer and Ronald Hecker for enabling the live demo of $\pi_{0.5}$ VLA model on the Trossen Robot.

References

B. Ai, S. Tian, H. Shi, Y. Wang, T. Pfaff, C. Tan, H. I. Christensen, H. Su, J. Wu, and Y. Li. A review of learning-based dynamics models for robotic manipulation. *Science Robotics*, 10(106):eadt1497, 2025.

S. Bai, W. Song, J. Chen, Y. Ji, Z. Zhong, J. Yang, H. Zhao, W. Zhou, W. Zhao, Z. Li, et al. Towards a unified understanding of robot manipulation: A comprehensive survey. *arXiv preprint arXiv:2510.10903*, 2025.

L. Beyer, A. Steiner, A. S. Pinto, A. Kolesnikov, X. Wang, D. Salz, M. Neumann, I. Alabdulmohsin, M. Tschannen, E. Bugliarello, et al. Paligemma: A versatile 3b vlm for transfer. *arXiv preprint arXiv:2407.07726*, 2024.

K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, et al. π_0: A vision-language-action flow model for general robot control. *arXiv preprint arXiv:2410.24164*, 2024.

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

Y. Hu, Q. Xie, V. Jain, J. Francis, J. Patrikar, N. Keetha, S. Kim, Y. Xie, T. Zhang, H.-S. Fang, S. Zhao, S. Omidshafiei, D.-K. Kim, A. akbar Agha-mohammadi, K. Sycara, M. Johnson-Roberson, D. Batra, X. Wang, S. Scherer, C. Wang, Z. Kira, F. Xia, and Y. Bisk. Toward general-purpose robots via foundation models: A survey and meta-analysis. *arXiv preprint arXiv:2312.08782*, 2023.

Intel Pytorch. Intel GPU get started. PyTorch Documentation. URL https://docs.pytorch.org/docs/2.12/notes/get_start_xpu.html.

D.-S. Jang, D.-H. Cho, W.-C. Lee, S.-K. Ryu, B. Jeong, M. Hong, M. Jung, M. Kim, M. Lee, S. Lee, et al. Unlocking robotic autonomy: A survey on the applications of foundation models. *International Journal of Control, Automation and Systems*, 22(8):2341–2384, 2024.

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

K. Kawaharazuka, J. Oh, J. Yamada, I. Posner, and Y. Zhu. Vision-language-action models for robotics: A review towards real-world applications. *IEEE Access*, 13:162467–162504, 2025. doi: 10.1109/ACCESS.2025.3609980. URL https://vla-survey.github.io/.

J. Luo, X. Zhou, C. Zeng, Y. Jiang, W. Qi, K. Xiang, M. Pang, and B. Tang. Robotics perception and control: Key technologies and applications. *Micromachines*, 15(4):531, 2024.

OepnVINO Developer Guide. OpenVINO model optimizer developer guide. Intel OpenVINO Toolkit. URL https://docs.openvino.ai/archives/index.html.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner,

L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019. URL https://arxiv.org/abs/1912.01703.

Physical Intelligence LIBERO. LIBERO benchmark. GitHub. URL https://github.com/Physical-Intelligence/openpi/tree/main/examples/libero.

G. R. Team, S. Abeyruwan, J. Ainslie, J.-B. Alayrac, M. G. Arenas, T. Armstrong, A. Balakrishna, R. Baruch, M. Bauza, M. Blokzijl, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. Sigmoid loss for language image pre-training. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11975–11986, 2023.

Notices and Disclaimers