Reconfigurable Computing Challenge: Deploying SmolVLA Model on an AMD XDNA NPU

Xiaoyu Liang, Elaine Cao, Chun-Ning Kao, David Lin, Jiajie Li, Zhiru Zhang
School of ECE, Cornell University, Ithaca, New York, USA

***Abstract*—Vision–Language–Action (VLA) models have demonstrated strong generalization across robotic tasks and embodiments. However, current deployments rely on GPU-based inference that is power inefficient for edge platforms, motivating the exploration of more efficient hardware targets. We present an initial exploration of the deployment of a VLA model on an AMD XDNA1 NPU, targeting SmolVLA and using Allo, a Python-based dataflow DSL. Operational intensity profiling identifies GEMM as the dominant kernel and the primary NPU target. To support SmolVLA's wide dynamic shape variation, we develop a pre-cached profiling database augmented by an LLM-assisted tiling agent. Normalization and activation kernels are further optimized through vectorization and approximations. This work results in a functionally correct end-to-end prototype. As an initial exploration, we also discuss current limitations and optimization opportunities toward practical VLA deployment on NPUs.**

I. Introduction

Modern robotics increasingly demands systems that can operate across diverse real-world environments, from assistive robots in healthcare [1], autonomous agents in warehouse logistics [2], to field robots in unstructured outdoor settings such as agricultural operations [3]. Traditional robot controllers struggle to meet this requirement: they are designed for fixed, short-horizon tasks, require extensive hand-engineering for each new scenario, and fail to generalize across the long-horizon, open-ended tasks that real-world deployment demands.

Recent advances in transformer-based vision and language models have enabled a new class of unified policies that can perceive the environment, understand human intent, and produce low-level motor commands within a single learned system. Vision–Language–Action (VLA) models represent a significant step toward this goal [4] [5] [6] [7]. By jointly training a large transformer on visual observations, natural-language instructions, and robot action sequences, VLA models learn generalizable sensorimotor policies that can be directed through language at runtime. This paradigm has demonstrated strong empirical results on various tasks, from pick-up and place of objects on a tabletop conditioned on language instructions [8], to dexterous bimanual manipulation such as folding laundry and packing groceries [9], to long-horizon mobile manipulation in unseen household environments [6].

These capabilities, however, come at a steep computational cost. Real-time robot control requires inference rates of at least 10 Hz, with high-frequency dexterous tasks demanding 50–100 Hz for smooth, reactive control. Although GPUs are the dominant deployment platform for VLA inference today, their power consumption, physical size, and thermal requirements make them unsustainable for smaller-scale mobile robots that must operate on battery power for extended periods. Meanwhile, the deployment of VLA models on compact, power-efficient edge hardware remains largely unexplored.

To close this gap, we explore the deployment of VLA on the AMD Ryzen AI XDNA1 NPU [10]. The XDNA1 integrates a 2D array of AI Engine (AIE) tiles into the same SoC as the CPU, delivering TOPS-class matrix throughput at single-digit watts. Each AIE compute tile contains a VLIW processor with a 512-bit SIMD vector datapath, 64 KB of local data memory, and a hardware streaming interconnect well-suited to the dataflows characteristic of transformer inference. We target SmolVLA [11], a compact 450 M-parameter VLA model designed for deployability and small enough to run on a consumer CPU, making it a realistic candidate for edge NPU deployment. Using Allo [12] [13], a Python-embedded DSL for dataflow accelerators, we map all major SmolVLA kernels to AIE tiles through tensor-level specifications that compile to complete tile programs via an MLIR backend. This approach substantially reduces the engineering overhead compared to low-level frameworks such as IRON [14]. This work makes the following contributions:

- **End-to-end functional prototype**: the first complete, numerically verified SmolVLA inference pipeline running on an NPU.
- **Dynamic GEMM optimization**: a pre-cached profiling database and an LLM-assisted tiling agent that achieves 58–68% exact match with empirically optimal tile configurations across SmolVLA's wide shape space.
- **Kernel optimization**: vectorized polynomial approximations for normalization and activation kernels, yielding 8–54× speedups over scalar baselines on XDNA1 AIE.

We further analyze the dispatch-overhead bottleneck of the initial prototype, describe the optimizations that close it, and outline remaining directions for future work.

II. Background

A. SmolVLA Architecture

SmolVLA [11] [15] is built around three transformer modules. The **Vision Encoder** (SigLIP ViT, 93 M parameters) processes camera images through a 12-layer ViT after patch splitting and Pixel Shuffle compression that trades spatial resolution for channel depth, reducing downstream sequence length while preserving information density. The **Text Encoder** (SmolLM-2, 250 M parameters) is a truncated LLaMA-2

transformer fusing visual tokens with the tokenized instruction, using RMS Norm, Grouped Query Attention (GQA) with RoPE, and SwiGLU feed-forward blocks. The **Action Expert** (100 M parameters) generates a sequence of action chunks via flow matching. Rather than waiting for the Text Encoder to complete its full forward pass, the Action Expert performs early cross-attention to intermediate (not final) Text Encoder features through interleaved self- and cross-attention layers, reducing action generation latency.

B. AMD XDNA NPU and Allo

The AMD Ryzen 9 7940HS integrates the XDNA1 NPU alongside 8 Zen4 CPU cores within the same SoC. XDNA1 follows a spatial dataflow architecture organized as a 4×5 grid of AI Engine (AIE) compute tiles, memory tiles, and interface tiles. Each AIE compute tile contains a VLIW processor with a 512-bit vector MAC unit, 64 KB of local data memory, and hardware locks for double-buffered synchronization. Tiles communicate through circuit-switched streaming interconnects that support a range of pipelined dataflows, in which selected operands or partial sums remain resident in local tile memory while others stream through the array. This dataflow model is particularly well-suited for GEMM-dominated workloads such as transformer inference, where large matrix multiplications can be decomposed into sub-tile products that stream through the array while accumulators remain on-chip, minimizing costly off-chip memory traffic.

Allo [12] [13] is a Python-embedded programming model for dataflow accelerators that has recently been extended to support AMD AIE hardware. Given a tensor-level operator specification, Allo lowers it to MLIR-AIE tile programs through three programmer-facing abstractions: *Mapping* assigns tasks to specific AIE compute tiles; *Layout* specifies how input tensors are sharded across those tiles; and *Streaming* defines on-chip point-to-point communication channels between tasks. We choose Allo over alternative NPU programming frameworks because lower-level interfaces such as IRON require manually specifying tile placement, DMA descriptors, and hardware lock sequences for every kernel, which are feasible for a single operator but impractical at the scale of a full VLA model. Allo instead exposes a Python-level tensor abstraction that compiles to correct AIE implementations automatically, enabling rapid iteration across all SmolVLA operators. Its primary limitation is that each kernel is compiled to an independent C++ host binary, incurring per-call XRT re-initialization overhead that we analyze and address in Section VI.

III. Workload Characterization

We analyze SmolVLA's computational structure to guide our NPU mapping decisions. Table I reports OI values derived analytically from model architecture and tensor shapes.

The analysis shows that GEMM operations are strongly compute-bound, with Vision Encoder GEMMs reaching 140–192 FLOPs/Byte, well above the CPU roofline. Activations and element-wise operations such as softmax and normalization have OI near or below 2, indicating they are memory-bandwidth-bound. The Vision Encoder GEMMs are more compute-intensive than those in the Action Expert due to larger token counts. This analytical observation is confirmed by CPU profiling on the Ryzen 9 7940HS using PyTorch's operator-level profiler: GEMM operations account for 66% of total inference time, with normalization (10%), positional embeddings (9%), data movement (6%), activations (5%), and softmax (4%) sharing the remainder, thereby making GEMM the primary focus of our NPU mapping effort.

TABLE I
Operational intensity of SmolVLA kernels

Kernel	FLOPs	Mem.	OI
GEMM – Vision Encoder	1.2–4.8 G	10–25 M	140–192
GEMM – Action Expert	23–148 M	1–6.5 M	20
Softmax	17.5 K	15 K	1.2
Normalization (LN / RMS)	180 K	250 K	0.7
Activations (SiLU/GeLU)	0.82–25 M	0.4–12 M	2
Full inference	250 G	3.5 G	70

IV. GEMM Optimization

A. GEMM Implementation on AIE

A GEMM computes $C = A \times B$, where $A \in \mathbb{R}^{M\times K}$, $B \in \mathbb{R}^{K\times N}$, and $C \in \mathbb{R}^{M\times N}$. To map this to the AIE array, we partition the computation into a logical $P_k \times P_m \times P_n$ tile grid, where $P_m = M/m$, $P_n = N/n$, and $P_k = K/k$ for tile sizes (m, n, k). Spatial parallelism is exploited along the M and N dimensions: $P_m \times P_n$ physical AIE tiles execute concurrently, each responsible for a distinct $m \times n$ output sub-tile. The K dimension is reduced sequentially, accumulating P_k partial products into a local BF16 buffer. This temporal K-reduction keeps all intermediate partial sums on-chip and avoids any inter-tile communication along the K dimension.

The choice of tile sizes (m, n, k) is subject to a memory constraint: the input sub-tiles $A[m \times k]$ and $B[k \times n]$ in BF16, together with the FP32 accumulator $C[m \times n]$, must all fit within the tile's local data store, while each dimension must be a multiple of the AIE-ML hardware matrix-multiply sub-tile dimensions. Within these constraints, larger tiles improve arithmetic intensity by amortizing DMA setup cost over more computation, but reduce the number of concurrent tiles and limit parallelism. Selecting the optimal valid (m, n, k) for a given (M, N, K) shape is therefore a non-trivial optimization problem, which is further complicated by SmolVLA's wide dynamic shape variation across model components.

B. Tiling Configuration Selection

SmolVLA's GEMM shapes vary substantially across model components, spanning M=1–1024, K=32–3072, N=32–2560 overall. The valid set of (m, n, k) configurations for any given (M, N, K) is small and irregular, and the best-performing configuration depends on shape in non-obvious ways, making exhaustive search at inference time impractical. Figure 1 illustrates this irregularity: at fixed M=256, the optimal tiling

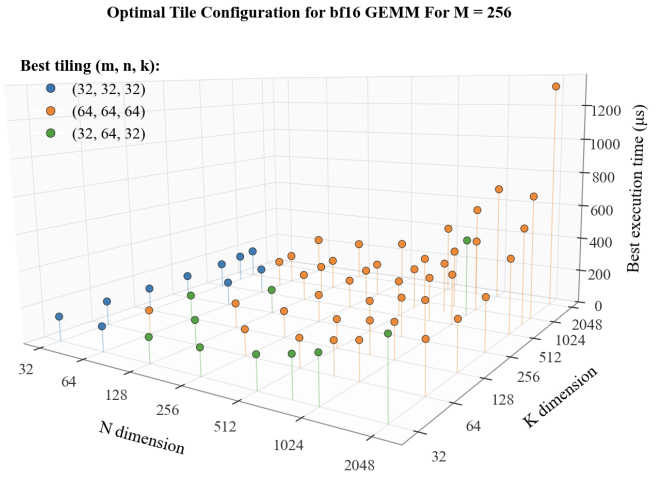

Fig. 1. Empirically optimal tile configuration (m, n, k) across (N, K) at fixed $M = 256$. The winning configuration switches non-monotonically among $(32, 32, 32)$, $(32, 64, 32)$, and $(64, 64, 64)$ with no clean closed-form rule, motivating a data-driven selection strategy. Similar irregularity is observed across other M values.

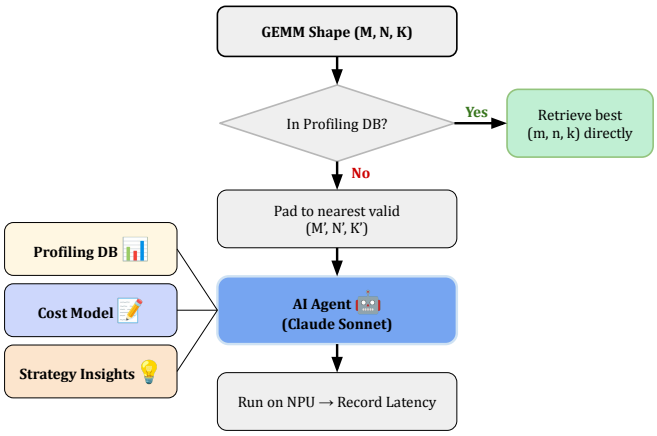

Fig. 2. Agentic tiling exploration. For unseen GEMM shapes, the agent proposes candidate (m, n, k) configurations given the profiling database, cost model, and strategy insights, benchmarks them on the NPU, and caches results for future reuse.

switches non-monotonically among $(32, 32, 32)$, $(32, 64, 32)$, and $(64, 64, 64)$ as (N, K) vary.

We address this with a unified approach centered on a growing profiling database. Offline, we enumerate common power-of-two (M, N, K) sizes, compile every valid tiling, run each on the NPU hardware, and record the best-performing (m, n, k). At inference time, shapes with a cached entry retrieve their optimal configuration instantly.

For shapes not present in the database, we deploy an LLM-based agent (Figure 2) that proposes candidate tile configurations given the profiling database, an analytic cost model, and hand-written architecture insights about AIE local memory and vector lane-width constraints. The agent's proposals are compiled and benchmarked on the NPU, and the best result is added to the database for future reuse. We evaluated this system on 50 held-out shapes with no ground-truth provided, comparing each recommendation against an exhaustive offline sweep. The agent matches the optimal tile 58–68% of the time with no search, and when it misses the performance penalty is small (3–13%), as nearby configurations tend to perform similarly. Notably, 13–27% of "mismatches" are actually faster than the labeled ground-truth best, suggesting the agent occasionally identifies configurations the finite offline sweep overlooked.

Together, the profiling database and agent form a complementary system where the database provides optimal configurations for known shapes and the agent generalizes efficiently to unseen ones.

V. Kernel Implementation and Optimization

For each SmolVLA kernel, we implement the single-tile compute kernel in C++ using the AIE API [16] (`aie::vector` intrinsics), and use Allo to specify the mapping to the AIE grid, memory layout, and inter-tile streaming for the full operator. All kernels are numerically verified against PyTorch CPU references at L_∞ tolerance 10^{-3}, appropriate for BF16 precision. We implement all kernels used in SmolVLA, including Conv2d, Layer Norm, RMS Norm, GeLU, SiLU, Softmax, and Sine/Cosine (for RoPE).

Our initial implementations of transcendental functions (GeLU, SiLU, sine, cosine) used scalar computation with lookup tables (LUTs). On AIE, LUTs require data-dependent memory accesses — each element indexes a different memory location based on its value, which the SIMD load unit cannot fully parallelize. Vectorizing the scalar LUT implementation provides a modest initial speedup, but the LUT itself remains the bottleneck. We replace LUTs with polynomial approximations, which eliminate indirect memory accesses entirely and allow the vectorization factor to scale to 16 or 32. Table II summarizes the resulting speedups, which range from 8–54× over the scalar LUT baseline.

TABLE II
Element-wise kernel optimization results on XDNA1 AIE.

Kernel	Method	Vec Factor	Speedup
GeLU	Rational approx. for tanh	32	∼54×
SiLU	Taylor e^{-x} (17 terms)	16	∼31×
Sine	Degree-3 Taylor + range reduce	16	∼21×
Cosine	Degree-3 Taylor + range reduce	16	∼23×
Softmax	Vectorization	16	∼8×
RMSNorm	Vectorization	32	∼12×
LayerNorm	Vectorization	16	∼12×

*Speedups are compared against scalar LUT baseline.

VI. End-to-End Prototype

A. End-to-End Result

Our complete SmolVLA prototype runs end-to-end on the XDNA1 NPU with all major transformer blocks executing on AIE tiles, and CPU fallbacks for the flow-matching action head. We verify functional correctness through the PushT

simulation [17], a standard task in which the robot end-effector must push a T-shaped object to a target position by controlling movement in the x and y directions. Our prototype produces correct action sequences and the robot successfully completes the task.

Table III reports end-to-end latency for the initial NPU prototype and improved results after each cumulative optimization stage. The initial attempt takes 654 s per inference, far slower than the 0.26 s CPU baseline. The key bottleneck is dispatch overhead, which we analyze briefly before describing the three optimizations that bring total latency down to 5 s.

TABLE III
SMOLVLA END-TO-END TIMING.

Stage	CPU	NPU	+ Fusion	+ Tiles	+ Unified
Preprocessing	0.088	453	8.6	8.6	0.16
Vision Encoder	0.055	50.2	50	17.8	3.5
Connector	0.008	95	20	19.9	0.18
Joint Transformer	0.109	56	25	12.8	1.59
Total	**0.26**	**654**	**104**	**59**	**5**

End-to-end wall-clock latency (seconds). Each + column applies the labeled optimization cumulatively to the previous configuration. This comparison reflects latency only and does not imply overall efficiency.

B. Dispatch Overhead Analysis

A single SmolVLA inference issues 21,783 separate NPU kernel dispatches, with preprocessing alone accounting for 14,592 (67%). Since Allo emits each kernel as a standalone C++ host binary that rebuilds the XRT stack on every call, each dispatch incurs roughly 28 ms of fixed overhead, of which $\sim$90% is XRT re-initialization (opening the device, loading the XCLBIN, creating a fresh execution context, re-allocating DMA buffers) rather than actual compute. The predicted cost of $21{,}783 \times 28\,\text{ms} \approx 610\,\text{s}$ closely matches the measured 654 s, confirming that dispatch overhead is the key bottleneck.

C. End-to-End Optimization

We apply three optimizations, evaluated cumulatively in Table III. End-to-end wall-clock time drops from 654 s to 5 s, a $130\times$ improvement that makes the NPU more feasible for smaller-scale tasks such as PushT.

Kernel fusion. We apply kernel fusion across preprocessing, the Connector, and the Joint Transformer to reduce dispatch count and keep intermediate tensors on-chip. The largest case rewrites preprocessing's per-patch im2col/slice/reshape chain as a two-phase im2col + GEMM pipeline. We similarly batch the Connector's Pixel-Shuffle and GEMM dispatches, and replace the joint transformer's RoPE chain with a single fused kernel using precomputed sin/cos tables.

Increased tile utilization. The element-wise and reduction kernels (LayerNorm, GELU, Softmax) were initially mapped to a small subset of compute tiles. We raise the parallel tile count for these non-GEMM kernels, for example, mapping Softmax across 16 AIE tiles instead of 4.

Unified C++ host. Allo generates a separate C++ host file for each kernel. By default the runtime launches these through Python, so every call spawns a fresh process and rebuilds the XRT context from scratch. We instead stitch the Allo-generated host files for all kernels of a stage into a single binary that acquires the XRT context once at startup and reuses it across every kernel call. The largest gains accrue in stages with many kernels (preprocessing, Connector), where per-call XRT setup was dominating.

VII. DISCUSSION

In this work, we present an initial exploration of deploying SmolVLA on the AMD XDNA1 NPU using Allo. We develop a dynamic GEMM tiling strategy that combines a pre-cached profiling database with an LLM-assisted tiling agent to handle SmolVLA's wide shape variation. We further optimize activation and normalization kernels through vectorized polynomial approximations, yielding 8–54$\times$ speedups over scalar baselines. Together these contributions result in a functionally correct end-to-end prototype, demonstrated through the PushT simulation task. Profiling identifies dispatch overhead as the dominant bottleneck of the initial prototype, motivating kernel fusion, increased tile utilization, and a unified C++ host that together reduce end-to-end NPU wall-clock time from 654 s to 5 s, a $130\times$ improvement that brings the NPU within range of practical use for smaller-scale tasks.

While this work focuses on SmolVLA, several compact VLA models have been proposed with recent work such as NanoVLA [18] and TinyVLA [19] targeting similar efficiency goals. Our approach is generalizable: the transformer kernels we implement are shared building blocks across most VLA architectures, and our dynamic GEMM tiling infrastructure is applicable to any model with variable sequence lengths or hidden dimensions. This suggests that the engineering effort explored here could transfer to other compact VLA models with relatively modest adaptation.

Looking ahead, the transformer blocks present additional kernel fusion opportunities that could further reduce dispatch count and improve data locality by keeping intermediate results on-chip. Exploiting the heterogeneity of the Ryzen AI SoC presents another opportunity. Mapping SmolVLA's three transformer components to the most suitable backend (CPU, NPU, or iGPU) could improve overall efficiency rather than forcing all components through a single target. Finally, migrating to XDNA2 could simplify kernel implementation and open new optimization opportunities, as the next-generation AIE architecture provides native hardware support for transcendental functions such as sine, cosine, and exponential that currently require custom implementation on XDNA1. We hope this work serves as a first step toward power-efficient VLA deployment on edge NPU platforms.

ACKNOWLEDGMENT

This work was supported in part by ACE, one of the seven centers in JUMP 2.0, a Semiconductor Research Corporation (SRC) program sponsored by DARPA, and a research gift from AMD. We also thank Shihan Fang for her support and valuable insights on Allo, and Qingyin Zhong for helping with the FCCM RCC demo.

REFERENCES

[1] X. Liang, Z. Liu, K. Lin, E. Gu, R. Ye, T. Nguyen, C. Hsu, Z. Wu, X. Yang, C. S. Y. Cheung, H. Soh, K. Dimitropoulou, and T. Bhattacharjee, "OpenRoboCare: A multi-modal multi-task expert demonstration dataset for robot caregiving," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2025.

[2] R. Keith and H. M. La, "Review of autonomous mobile robots for the warehouse environment," *arXiv preprint arXiv:2406.08333*, 2024.

[3] K. Truong, Y. Lee, J. Irie, S. K. Panda, M. Jony, S. Ahmad, M. M. Rahman, and M. K. Jawed, "Agricruiser: An open source agriculture robot for over-the-row navigation," 2025. [Online]. Available: https://arxiv.org/abs/2509.25056

[4] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn, "Openvla: An open-source vision-language-action model," 2024. [Online]. Available: https://arxiv.org/abs/2406.09246

[5] K. Black, N. Brown, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, L. X. Shi, J. Tanner, Q. Vuong, A. Walling, H. Wang, and U. Zhilinsky, "π_0: A vision-language-action flow model for general robot control," 2024. [Online]. Available: https://arxiv.org/abs/2410.24164

[6] P. Intelligence, K. Black, N. Brown, J. Darpinian, K. Dhabalia, D. Driess, A. Esmail, M. Equi, C. Finn, N. Fusai, M. Y. Galliker, D. Ghosh, L. Groom, K. Hausman, B. Ichter, S. Jakubczak, T. Jones, L. Ke, D. LeBlanc, S. Levine, A. Li-Bell, M. Mothukuri, S. Nair, K. Pertsch, A. Z. Ren, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, J. Tanner, Q. Vuong, H. Walke, A. Walling, H. Wang, L. Yu, and U. Zhilinsky, "$\pi_{0.5}$: a vision-language-action model with open-world generalization," 2025. [Online]. Available: https://arxiv.org/abs/2504.16054

[7] P. Intelligence, B. Ai, A. Amin, R. Aniceto, A. Balakrishna, G. Balke, K. Black, G. Bokinsky, S. Cao, T. Charbonnier, V. Choudhary, F. Collins, K. Conley, G. Connors, J. Darpinian, K. Dhabalia, M. Dhaka, J. DiCarlo, D. Driess, M. Equi, A. Esmail, Y. Fang, C. Finn, C. Glossop, T. Godden, I. Goryachev, L. Groom, H. Habeeb, H. Hancock, K. Hausman, G. Hussein, V. Hwang, B. Ichter, C. Jacobsen, S. Jakubczak, R. Jen, T. Jones, G. Kammerer, B. Katz, L. Ke, M. Khadikov, C. Kuchi, M. Lamb, D. LeBlanc, B. LeCount, S. Levine, X. Li, A. Li-Bell, V. Lialin, Z. Liang, W. Lim, Y. Lu, E. Luo, V. Mano, N. Marwaha, A. Mongush, L. Murphy, S. Nair, T. Patterson, K. Pertsch, A. Z. Ren, G. Schelske, C. Sharma, B. Shi, L. X. Shi, L. Smith, J. T. Springenberg, K. Stachowicz, W. Stoeckle, J. Tang, J. Tanner, S. Tekeste, M. Torne, K. Vedder, Q. Vuong, A. Walling, H. Wang, J. Wang, X. Wang, C. Whalen, S. Whitmore, B. Williams, C. Xu, S. Yoo, L. Yu, W. Zhang, Z. Zhang, and U. Zhilinsky, "$\pi_{0.7}$: a steerable generalist robotic foundation model with emergent capabilities," 2026. [Online]. Available: https://arxiv.org/abs/2604.15483

[8] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn, P. Florence, C. Fu, M. G. Arenas, K. Gopalakrishnan, K. Han, K. Hausman, A. Herzog, J. Hsu, B. Ichter, A. Irpan, N. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, I. Leal, L. Lee, T.-W. E. Lee, S. Levine, Y. Lu, H. Michalewski, I. Mordatch, K. Pertsch, K. Rao, K. Reymann, M. Ryoo, G. Salazar, P. Sanketi, P. Sermanet, J. Singh, A. Singh, R. Soricut, H. Tran, V. Vanhoucke, Q. Vuong, A. Wahid, S. Welker, P. Wohlhart, J. Wu, F. Xia, T. Xiao, P. Xu, S. Xu, T. Yu, and B. Zitkovich, "RT-2: Vision-language-action models transfer web knowledge to robotic control," in *Proceedings of The 7th Conference on Robot Learning (CoRL)*, 2023.

[9] S. Liu, L. Wu, B. Li, H. Tan, H. Chen, Z. Wang, K. Xu, H. Su, and J. Zhu, "Rdt-1b: a diffusion foundation model for bimanual manipulation," *arXiv preprint arXiv:2410.07864*, 2024.

[10] A. Rico, S. Pareek, J. Cabezas, D. Clarke, B. Ozgul, F. Barat, Y. Fu, S. Münz, D. Stuart, P. Schlangen, P. Duarte, S. Date, I. Paul, J. Weng, S. Santan, V. Kathail, A. Sirasao, and J. Noguera, "Amd xdna npu in ryzen ai processors," *IEEE Micro*, vol. 44, no. 6, pp. 73–82, 2024.

[11] M. Shukor, D. Aubakirova, F. Capuano, P. Kooijmans, S. Palma, A. Zouitine, M. Aractingi, C. Pascal, M. Russi, A. Marafioti, S. Alibert, M. Cord, T. Wolf, and R. Cadene, "Smolvla: A vision-language-action model for affordable and efficient robotics," 2025. [Online]. Available: https://arxiv.org/abs/2506.01844

[12] H. Chen, N. Zhang, S. Xiang, Z. Zeng, M. Dai, and Z. Zhang, "Allo: A programming model for composable accelerator design," *Proceedings of the ACM on Programming Languages*, vol. 8, no. PLDI, pp. 593–620, 2024.

[13] S. Fang, H. Chen, N. Zhang, J. Li, H. Meng, A. Liu, and Z. Zhang, "Dato: A task-based programming model for dataflow accelerators," *arXiv preprint arXiv:2509.06794*, 2025.

[14] E. Hunhoff, J. Melber, K. Denolf, A. Bisca, S. Bayliss, S. Neuendorffer, J. Fifield, J. Lo, P. Vasireddy, P. James-Roxby, and E. Keller, "Efficiency, expressivity, and extensibility in a close-to-metal npu programming interface," 2025. [Online]. Available: https://arxiv.org/abs/2504.18430

[15] A. Marafioti, O. Zohar, M. Farré, M. Noyan, E. Bakouch, P. Cuenca, C. Zakka, L. B. Allal, A. Lozhkov, N. Tazi, V. Srivastav, J. Lochner, H. Larcher, M. Morlon, L. Tunstall, L. von Werra, and T. Wolf, "Smolvlm: Redefining small and efficient multimodal models," 2025. [Online]. Available: https://arxiv.org/abs/2504.05299

[16] *AI Engine API User Guide (AIE)*, v2022.2 ed., AMD, 2022, accessed: May 21, 2026. [Online]. Available: https://download.amd.com/docnav/aiengine/xilinx2022_2/aiengine_api/aie_api/doc/index.html

[17] Hugging Face, "gym-pusht: A gym environment for PushT," https://github.com/huggingface/gym-pusht, 2024, accessed: May 21, 2026.

[18] Y. Wen *et al.*, "Nanovla: Routing decoupled vision-language understanding for nano-sized generalist robotic policies," *arXiv preprint arXiv:2510.25122*, 2025.

[19] J. Wen, Y. Zhu, M. Zhu, J. Li, Z. Xu, N. Liu, C. Shen, Y. Peng, F. Feng, S. Zheng, and J. Liu, "Tinyvla: Towards fast, data-efficient vision-language-action models for robotic manipulation," *arXiv preprint arXiv:2409.12514*, 2024.