TOWARD A SMALL ML RUNTIME STACK FOR RASPBERRY PI 5 QPUS

Yiannis Hadjiyianni [1] **Panagiotis Michelakis** [1 †] **Dimitrios Stamoulis** [2]

ABSTRACT

We present a QPU-*first* ML runtime stack for Raspberry Pi 5's VideoCore VII QPU, built on top of the `py-videocore7` assembly library[1]. The system comprises reusable tiled matrix-multiplication substrate, GEMM-backed convolution, a single-head attention-style core, persistent executors, and integer execution based on `smul24` instructions. For dense integer kernels, packed INT16-input with INT32 accumulation achieves nearly two orders of magnitude higher throughput over NumPy. Across operations (min/max, pooling, convolution, attention), we report improved performance over both PyTorch and NumPy. Our preliminary results indicate that Raspberry QPUs can serve as a practical execution substrate towards accelerating AI model execution at the edge.

Raspberry Pi is widely deployed in edge and IoT systems (Panopoulos et al., 2026), yet its integrated QPU is rarely treated as an ML execution device. To our knowledge, there is no end-to-end runtime stack for QPUs: all existing inference engines (*e.g.*, ONNX (ONNX, 2026), Ollama (Ollama, 2026)) target its ARM CPU, leaving a gap between low-level QPU programmability and reusable inference-runtime support (Ardakani et al., 2025; Wei et al., 2025). This undergraduate student-led work examines whether QPUs can support structured runtime execution.

Methodology. We build on `pyvideocore7` (Idein, 2026), which provides Python-side *assembly* generation, kernel loading, and dispatch for VideoCore VII QPUs. We implement **custom** QPU kernels, a tiled execution substrate, persistent executors, and an operator layer. The core design choice is to reuse a tiled matrix-multiplication substrate across multiple workloads. Convolution is implemented as GEMM-backed convolution through lowering, while the attention operator is a single-head dot-product attention core of the form $O = (QK^\top)V$. We build persistent executors that assemble kernels and allocate device buffers once; our cache uniforms and dispatch metadata for repeated execution. Moreover, we optimize for integer execution rather than using generic tensor-accelerator (Torch) semantics: we implement integer kernels that conform to VideoCore arithmetic constraints induced by the `smul24` instruction. These operand-range contracts propagate to higher-level operators to determine feasible tiling, packing, and accumulation at runtime. Our evaluation therefore reports both operand-level throughput and operator-level runtime, while separating one-time setup, cached steady-state execution (QPU-C), and execute-only (QPU-E) kernel time.

Results. Table 1 summarizes dense integer GEMM performance. Standard INT32 GEMM peaks at 20.49 GOPS, while packed INT16-input with INT32 accumulation reaches 21.67 GOPS, with up to 94.38× speedup over NumPy. Table 2 reports operator-level performance. Integer operators show strongest gains, *e.g.*, INT32 avgpool reaches 2.62 GiB/s above both NumPy (0.72 GiB/s) and PyTorch (0.69 GiB/s). INT32 GEMM-backed convolution reaches 15.08 and 18.60 GOPS for QPU-C and QPU-E, respectively. For the INT32 attention, throughput is 12.81 GOPS for QPU-C *vs*. 14.41 GOPS for QPU-E. The gap between cached and execute-only is small once setup is amortized, supporting a runtime design with persistent executors and precompiled kernels. **CNN performance:** In preliminary experimentation, we have successfully run an end-to-end QPU-first *LeNet* model, where the 12-core execute-only INT32 pipeline reaches 4.08 GOPS, compared to 0.83 GOPS for NumPy, a nearly 5× performance increase.

Discussion. This work-in-progress investigation established a technically meaningful execution substrate on VideoCore VII: a reusable tiled backbone, operator implementations that inherit its layout and dispatch strategy, explicit integer contracts tied to the underlying machine, and a measurement methodology that distinguishes compilation, host orchestration, and steady-state device execution. In our ongoing work (Michelakis et al., 2025), we aim towards a full production backend, with broader operator coverage, CPU–QPU scheduling, and lightweight end-to-end LLM model inference on Raspberry Pi.

[†]Currently with new affiliation. [1]School of Electrical and Computer Engineering, National Technical University of Athens, Athens, Greece [2]Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, China. Correspondence to: Yiannis Hadjiyianni <yiannisha@synkrasis-labs.com>.

MLSys 2026 Young Professionals Symposium (YPS), Bellevue, WA, USA, 2026.

Table 1. Integer GEMM throughput (GOPS).

Size	INT32 GEMM			Packed INT16-Input/INT32-Accumulation		
	NumPy	QPU	Speedup	NumPy	QPU	Speedup
256	1.18	6.30	5.34×	1.17	6.30	5.40×
512	0.57	16.69	29.49×	0.57	16.82	29.70×
768	0.57	20.49	36.25×	0.60	20.49	33.91×
1024	0.21	10.73	50.15×	0.23	21.67	94.38×

Table 2. Operator-level Raspberry PI performance.

Operator[¶]	Setting	NumPy	PyTorch	QPU-C[*] Steady-State (Cached)	QPU-E Execute-Only
Min/max	INT32, 12 cores	6.27	6.17	**6.50**	—
Min/max	INT16, 12 cores	**6.71**	6.26	6.02	—
AvgPool 2×2 / 2	INT32, 12 cores	0.72	0.69	**2.62**	—
MaxPool 2×2 / 2	FP32, 12 cores	2.66	**3.03**	2.65	—
Conv2D	FP32	**21.69**	12.74	16.25	19.87
Conv2D	INT32	1.44	8.30	**15.08**	18.60
Conv2D	INT16-IN/INT32-ACC	1.45	15.45[†]	**15.97**	18.43
Attention	Core total, FP32	**82.35**	5.97[‡]	12.74	14.31
Attention	Core total, INT32	1.13	1.79	**12.81**	14.41

[¶] Measured as min/max, pooling: GiB/s; convolution, attention: GOPS.
[*] Cached: repeated execution w/ steady-state kernels and buffers w/o one-time setup.
[†] Native PyTorch INT16 convolution is used as a speed baseline only on this build.
[‡] The FP32 attention baseline is native SDPA and softmax with scale 1.0.

REFERENCES

Ardakani, M., Malekar, J., and Zand, R. Llmpi: Optimizing llms for high-throughput on raspberry pi. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 6369–6378, 2025.

Idein. py-videocore7: A python library for GPGPU programming on Raspberry Pi 5. https://github.com/Idein/py-videocore7, 2026. GitHub.

Michelakis, P., Hadjiyianni, Y., and Stamoulis, D. Core: Full-path evaluation of llm agents beyond final state, 2025. arXiv:2509.20998.

Ollama. Ollama: Start building with open models. https://github.com/ollama/ollama, 2026. GitHub.

ONNX. ONNX: Open neural network exchange. https://github.com/onnx, 2026. GitHub.

Panopoulos, I., Bartsioka, M. L. A., Nikolaidis, S., Venieris, S. I., Kaklamani, D. I., and Venieris, I. S. A-thena: Early intrusion detection for iot with time-aware hybrid encoding and network-specific augmentation. *ACM Transactions on AI Security and Privacy*, 2026.

Wei, J., Cao, S., Cao, T., Ma, L., Wang, L., Zhang, Y., and Yang, M. T-mac: Cpu renaissance via table lookup for low-bit llm deployment on edge. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 278–292. ACM, March 2025.