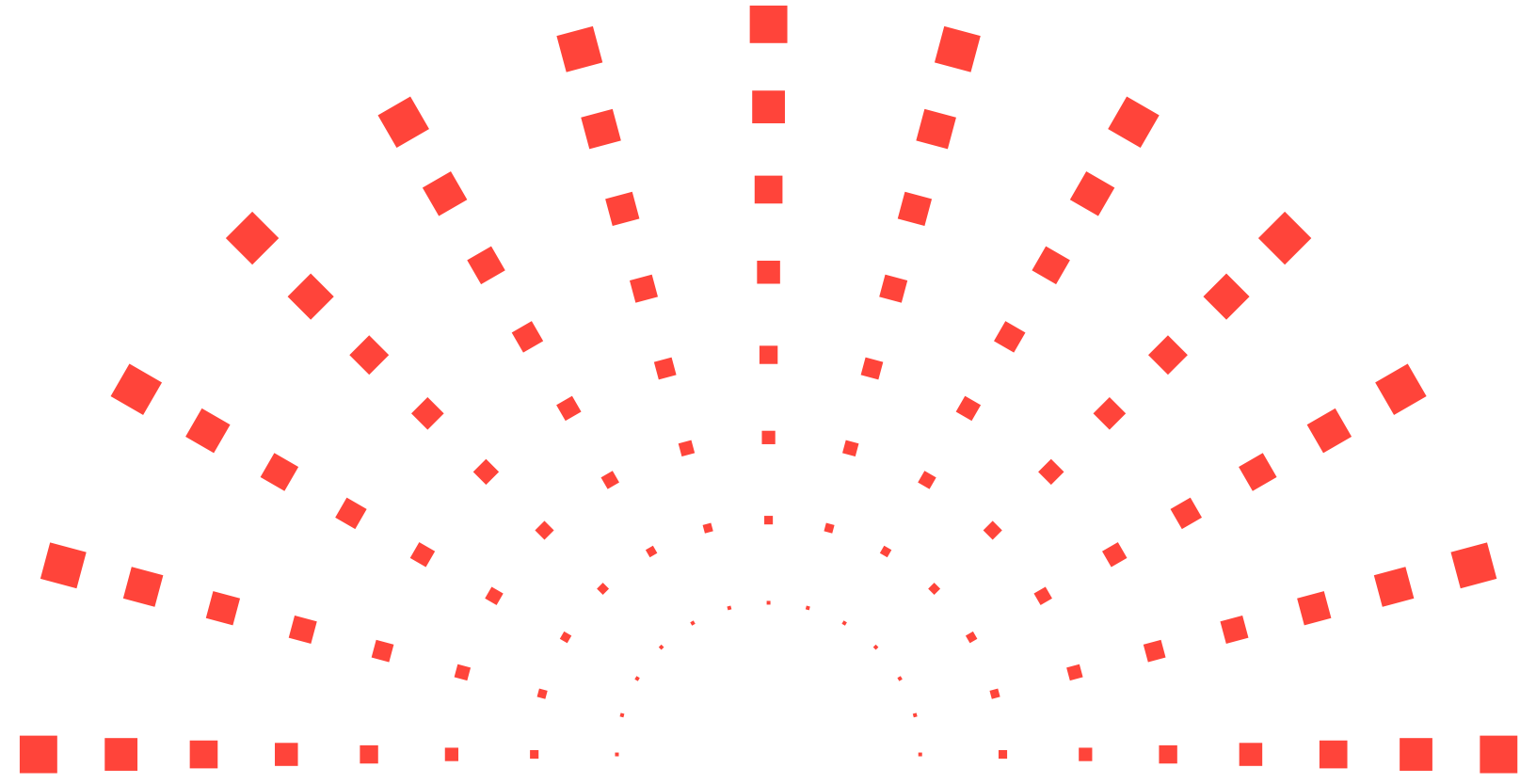
QNX | HAILO

White Paper

QNX for Deterministic Physical AI Workloads

A comparative benchmark of embedded NPU inference performance on Hailo-8



Abstract

With Physical AI becoming more prevalent in complex systems, Embedded NPUs (Neural Processing Unit) are moving into vehicles, robots, surgical equipment, factory machinery, and other devices where perception output can directly influence physical behavior. In these products, the average inference performance is not enough. Engineers and Product Designers also need consistency in timing so they can budget, validate, and defend their complex projects.

This paper compares the performance attributes of QNX SDP 8.x and PREEMPT_RT Linux running on identical hardware: a Raspberry Pi 5 with a Hailo-8 AI HAT+. The same C++ benchmark application, compiler version, MobileNet v1 model, HailoRT version, dataset, and cooling configuration were used on both platforms. Each operating system processed 100 consecutive runs of 10,000 frames, representing one million measured inferences per OS (Operating System).

Within this benchmark scope, QNX measured ahead of Linux across the reported performance and predictability metrics. QNX delivered 4.1% higher throughput, 3.9% lower end-to-end latency, 2.5% lower inference-only latency, 14 times tighter run-to-run FPS (frames per second) variation, 2.6 times lower per-frame latency variation and 35% fewer driver interactions per 10,000-frame run. These results indicate that the operating system is a material variable in the inference timing case for Physical AI devices.

Executive Summary

Core Findings for Physical AI

For Physical AI systems, the core finding is that QNX delivers higher throughput and delivers more predictable inference timing on the same hardware, model, dataset, and runtime. The predictability of inference especially matters when AI output drives physical action under real-time, safety, thermal, and certification constraints.

The Bottom Line

Using the same Raspberry Pi 5, Hailo-8, MobileNet v1 model, image set, compiler, HailoRT version and cooling configuration - QNX produced higher throughput, lower latency, and materially tighter timing consistency than PREEMPT_RT Linux.

What Was Measured?

The benchmark evaluates AI inference performance and timing consistency while changing only the operating system used to run the hardware. The hardware platform was a Raspberry Pi 5 with a Hailo-8 AI HAT+. The workload was a MobileNet v1 image classification model using 3,600 ImageNet-derived images.

Each platform ran 100 consecutive benchmark invocations of 10,000 frames each. The measurements captured throughput, end-to-end latency, inference-only latency, run-to-run variation, per-frame latency variation, CPU behavior, thermal behavior and control-plane driver costs.

The benchmark was intentionally synchronous, single-threaded, and single-frame. The design exposes scheduling, DMA (Direct Memory Access), interrupt delivery and driver behavior instead of hiding behind buffering or pipeline depth.

Key findings

Metric	QNX	Linux PREEMPT_RT	QNX result
Average throughput	681.07 FPS	654.23 FPS	+4.1%
Run-to-run FPS variation	σ = 0.22	σ = 3.06	14× tighter
End-to-end mean latency	1.468 ms	1.528 ms	3.9% lower
Inference-only mean latency	1.375 ms	1.410 ms	2.5% lower
Per-frame latency variation	9.21 µs	24.10 µs	2.6× tighter
Driver calls per 10,000-frame run	40,083	61,528	35% fewer
Average interrupt wait	743 µs	780 µs	4.7% lower
Steady-state Hailo-8 temperature	48.4 °C	50.3 °C	1.9 °C cooler

Why this Matters for Physical AI

Physical AI devices do not merely classify images or generate predictions; they take or inform action in the physical world. A mobile robot may choose a path, a surgical device may interpret anatomy, a vehicle may adjust behavior, and a factory controller may change a process. In each case, late or inconsistent inference can affect the timing budget for the rest of the system.

The QNX advantage in consistency is therefore more important than the throughput improvement alone. A 4.1% non-streaming FPS gain is valuable, but a 14× tighter run-to-run distribution and a 2.6× tighter per-frame distribution is what helps engineering teams bound worst-case behavior, size buffers, schedule tasks, validate system timing, and support safety or certification arguments.

The benchmark also shows that customers can keep NPU-accelerated inference inside a QNX-based architecture instead of adding a separate Linux environment only to access the accelerator. This matters for products that already rely on QNX for real-time behavior, isolation, reliability and safety-oriented engineering practices.

1. Introduction

AI inference is moving from cloud infrastructure into embedded devices at the edge. Vehicles, robotic systems, industrial machines, and medical devices increasingly need local perception because cloud round trips are too slow, too unreliable or difficult to certify for the control loop.

Dedicated NPUs such as the Hailo-8 provide the raw acceleration needed for efficient edge inference. However, silicon throughput alone does not determine whether an AI function can fit into a real-time product. The surrounding operating system must schedule the thread, manage DMA buffers, deliver interrupts, and return results predictably to complete the execution.

This paper compares two operating system architectures on the same NPU workload. QNX uses a microkernel architecture with user-space resource managers and priority-driven scheduling. PREEMPT_RT Linux uses a monolithic kernel with real-time patches, but still retains background kernel activity such as workqueues, RCU (Read-Copy-Update) callbacks, timer activity and threaded interrupt handling. The benchmark asks how much that difference matters in practice.

2. The Business Need

Many high-value embedded products already use QNX because they require deterministic behavior, long lifecycle support and a credible path to safety evidence. These products include automotive ECUs (Electronic Control Units), surgical platforms, factory vision systems, robotics platforms, and other safety-relevant edge devices.

At the same time, many accelerator software stacks are Linux-first or Linux-only. If a customer has committed to QNX for the control system, but the NPU software stack is only validated on Linux, the customer faces an unattractive choice: introduce Linux into a safety-relevant architecture, delay product plans while waiting for an RTOS port, or avoid the accelerator altogether.

The Hailo-8 integration demonstrates that QNX can support the NPU path directly. This benchmark then evaluates whether the QNX implementation merely works, or whether it can match and exceed the Linux baseline on performance and timing consistency.

3. The Benchmark Objective

The objective was to isolate the operating system as the only changing variable and measure its effect on NPU inference behavior. The study used the same hardware, same model, same dataset, same runtime version, same compiler version, same cooling configuration and the same benchmark application logic on both platforms.

The core metrics were non-streaming throughput, end-to-end latency, inference-only latency, run-to-run consistency, per-frame latency variation, and control-plane driver cost. The benchmark also reviewed CPU, memory, and thermal behavior to assess whether secondary effects explained the result.

4. Experimental Setup

Hardware Platform

Component	Specification
Board	Raspberry Pi 5
RAM	8 GB LPDDR4X
NPU	Hailo-8 AI HAT+ with 26 TOPS, PCIe interface
Cooling	External fan for ambient temperature control
Power supply	Official Raspberry Pi 5 PSU
Overclocking	Disabled

Software Configuration

Parameter	QNX	Linux
OS / kernel	QNX SDP 8.x, microkernel RTOS	Linux 6.12.62+rpt-rpi-v8-rt
Real-Time Support	Native RTOS scheduling	PREEMPT_RT
HailoRT	v4.14.0	v4.14.0
Compiler	qcc using GCC 12.2.0	aarch64-linux-gnu-g++ using GCC 12.2.0
Decode and resize	stb_image and custom ARM NEON resize	stb_image and custom ARM NEON resize
Real-time tuning	SCHED_FIFO priority 253, CPU 0 runmask, mlockall	SCHED_FIFO priority 99, CPU 0 affinity, mlockall
PAGE SIZE	getconf PAGESIZE=4096	getconf PAGESIZE=4096

Workload

Parameter	Value
Model	MobileNet v1, approximately 4.2 million parameters
Input dimensions	224 × 224 × 3, NHWC
Output classes	1,000 ImageNet classes
Model format	Hailo Executable Format (.hef)
Image dataset	3,600 ImageNet-derived JPEGs across seven source resolutions
Frames per invocation	10,000, with five warmup frames excluded from metrics
Number of invocations	100 consecutive runs per OS
Total measured scale	One million inferences per OS

5. Methodology

The benchmark application is a single C++ file compiled for both targets. The inference loop, preprocessing kernel, allocator pattern, and HailoRT API sequence are common across both builds. Platform-specific code is limited to timestamp and real-time tuning calls required by each OS.

The loop is deliberately synchronous: one frame is resized, written to the NPU input stream, processed by the NPU and read back before the next frame begins. A producer-consumer design could hide interrupt-wait variation behind buffering. This test exposes per-frame scheduling and driver behavior.

End-to-end latency measures the complete per-frame path, including NEON resize, NPU submission and result readback. Inference-only latency measures the write()+read() window, isolating the OS-mediated NPU round trip from CPU-side preprocessing.

6. Results and Analysis

6.1 Throughput

QNX delivered an average of 681.07 non streaming FPS versus 654.23 FPS for Linux, a 4.1% measured advantage. The distributions did not overlap: QNX's lowest reported run still exceeded Linux's highest reported run.

Neither figure is the Hailo-8's ceiling: with one frame in flight at a time, throughput here is decided by how fast each OS hands the NPU its next frame after the completion interrupt, and QNX consistently fires the next write() sooner. Driven inference-only and streaming, the same silicon delivers several thousand FPS on this model — Hailo's published Model Zoo figure for MobileNet v1 on Hailo-8 is 3,305 FPS.

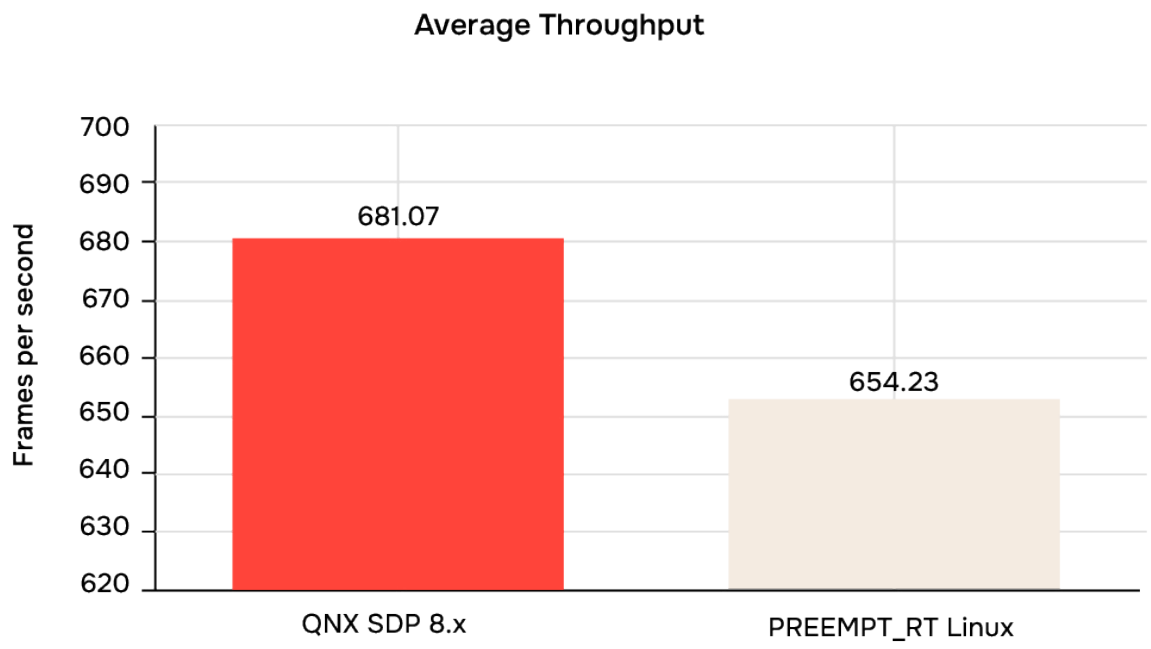


Figure 1. Average throughput. QNX delivered 4.1% higher non-streaming throughput on the same hardware and workload.

6.2 Latency

QNX delivered lower latency in both measured windows. End-to-end mean latency was 1.468 ms on QNX versus 1.528 ms on Linux. A lower latency number means the OS has better latency to execute on operations. Inference-only latency was 1.375 ms on QNX versus 1.410 ms on Linux.

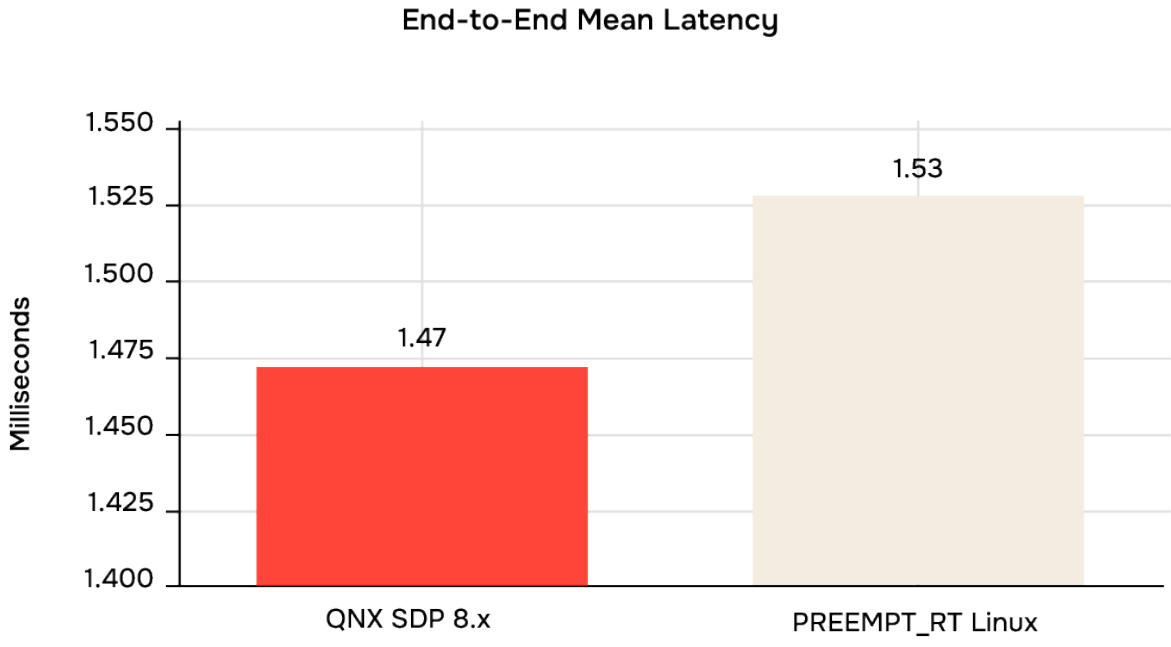


Figure 2. End-to-end mean latency. QNX delivered a 3.9% lower end-to-end latency result.

6.3 Determinism and Timing Consistency

The most important difference is timing consistency. QNX held run-to-run FPS variation to σ = 0.22, while Linux measured σ = 3.06, a 14× wider run-to-run spread on Linux. Within each run, per-frame latency variation averaged 9.21 µs on QNX versus 24.10 µs on Linux, making QNX 2.6× tighter.

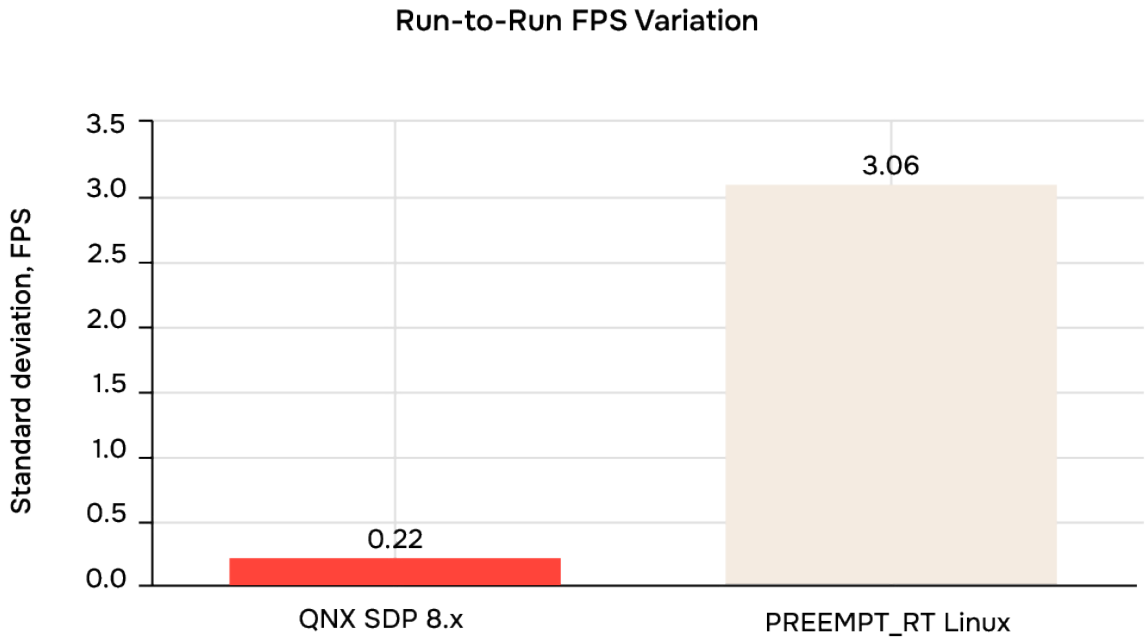


Figure 3. Run-to-run FPS variation. QNX produced a much tighter distribution across 100 consecutive runs.

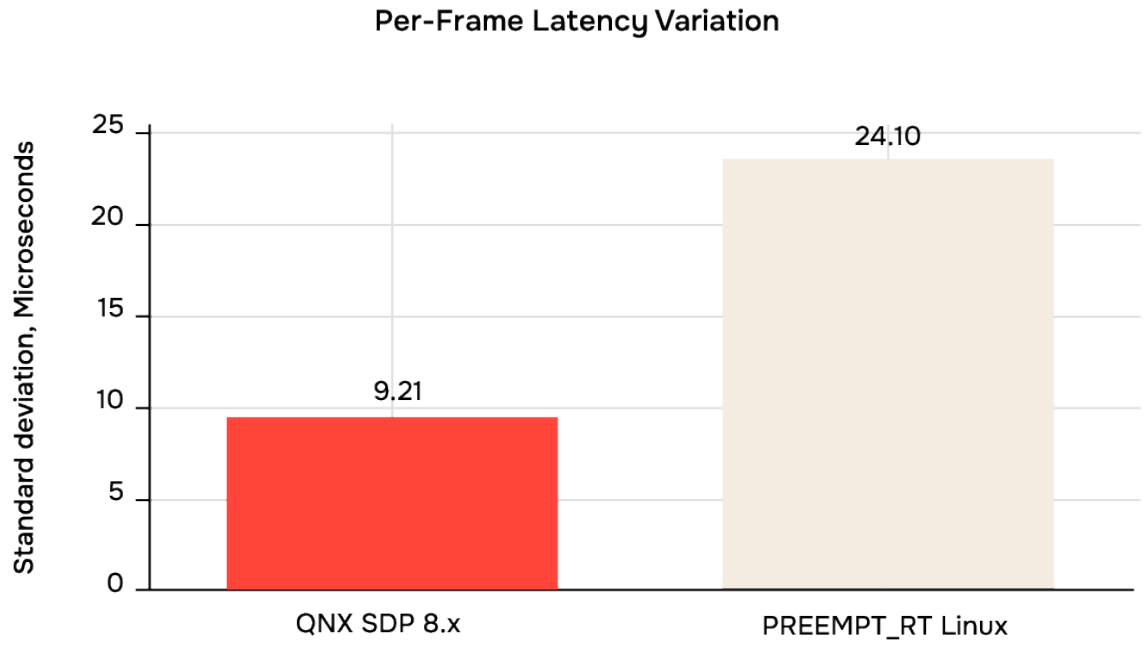


Figure 4. Per-frame latency variation. QNX produced a tighter frame-to-frame timing profile.

6.4 Control-plane Driver Cost

The driver traces indicate that Linux executed 61,528 driver calls per 10,000-frame run, while QNX executed 40,083. The Linux total includes 21,441 VDMA_BUFFER_SYNC calls with no QNX equivalent in this benchmark. QNX also reported a lower average interrupt wait time: 743 µs versus 780 µs on Linux. QNX achieved higher AI throughput with 35% fewer driver interactions, indicating a more efficient software path between the application and accelerator hardware.

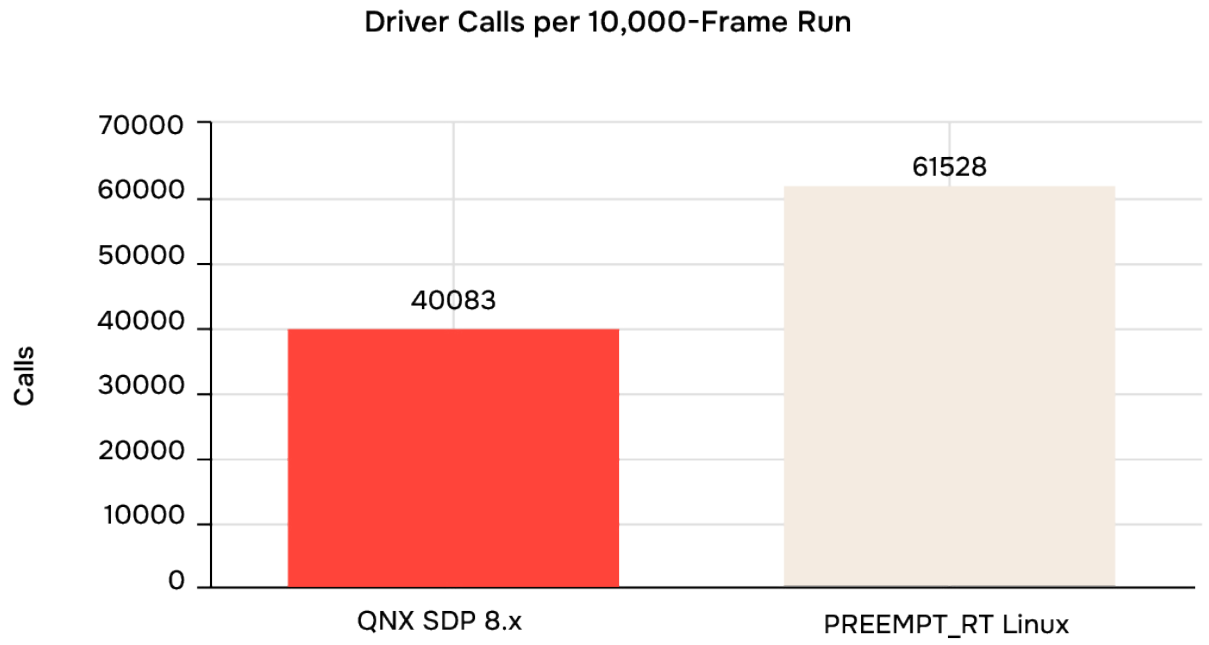
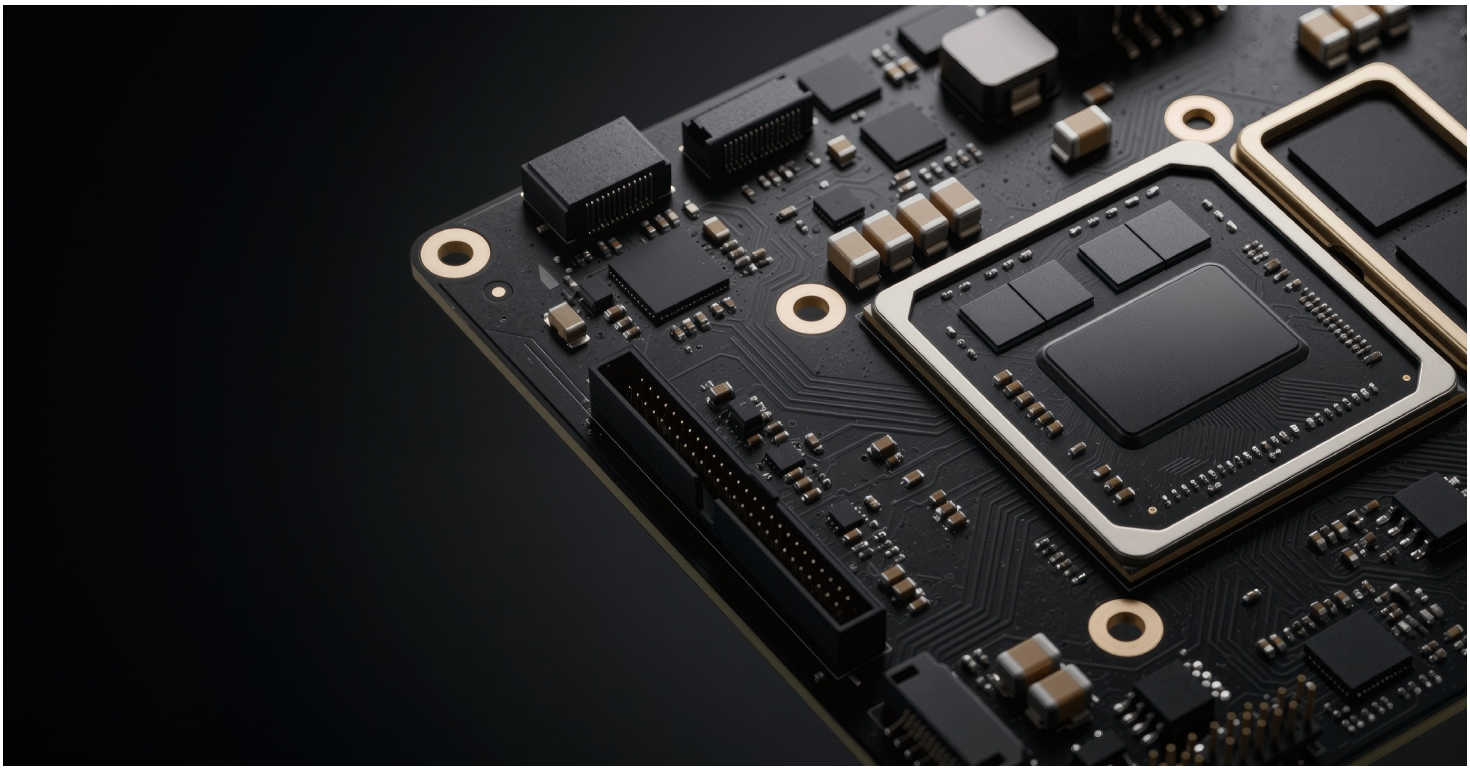


Figure 5. Driver call count. QNX required 35% fewer driver interactions per 10,000-frame run.



6.5 Thermal Behavior

Both platforms remained thermally stable under the external fan. QNX reported a lower steady-state Hailo-8 temperature of 48.4 ˚C versus 50.3 ˚C on Linux. This means QNX delivered higher throughput while leaving more thermal headroom in this configuration.

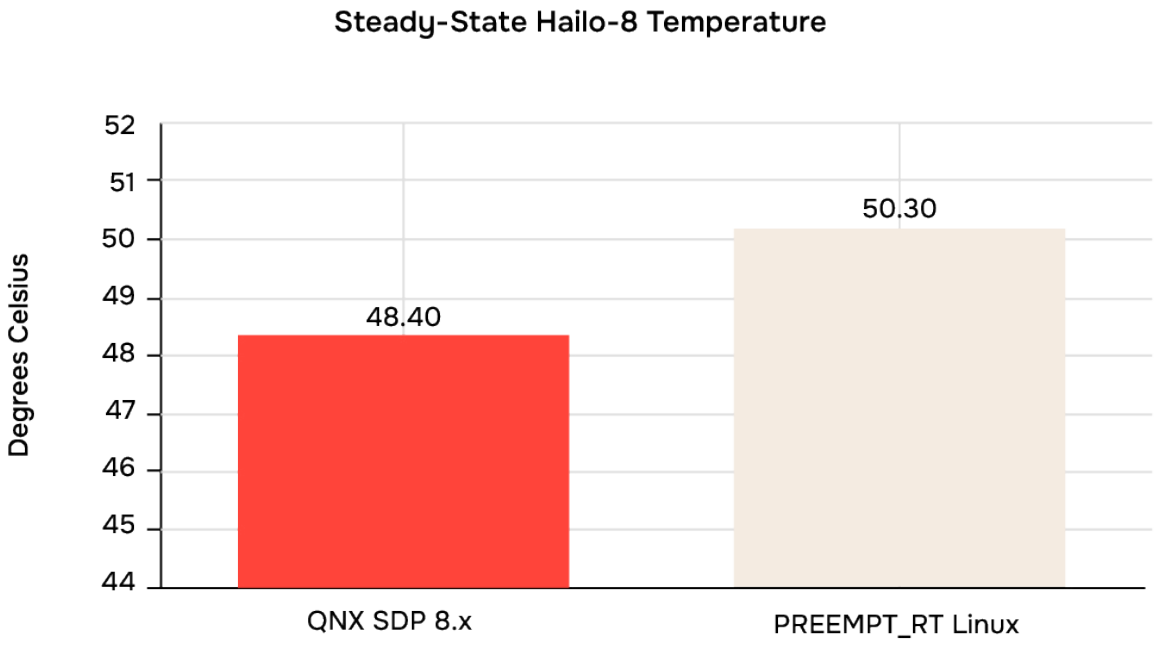


Figure 6. Steady-state Hailo-8 temperature. QNX ran approximately 1.9 ˚C cooler in the benchmark.

6.6 Consolidated KPI Comparison

KPI	QNX	Linux	Observed QNX advantage
Average FPS	681.07 ± 0.22	654.23 ± 3.06	+4.1%
Run-to-run FPS variation	0.22	3.06	14× tighter
End-to-end mean latency	1.468 ms	1.528 ms	3.9% lower
Inference-only mean latency	1.375 ms	1.410 ms	2.5% lower
Per-frame standard deviation	9.21 µs	24.10 µs	2.6× tighter
Driver calls per run	40,083	61,528	35% fewer
Interrupt wait	743 µs	780 µs	4.7% lower
Descriptor-list create	57 µs	23,253 µs	408× faster

7. Discussion: Why the OS Matters

The results are directionally consistent across throughput, latency, jitter, driver call count, and temperature. Because the hardware, model, dataset, compiler and benchmark application were held constant, the measured differences point to OS-level scheduling, interrupt delivery and DMA-management behavior as the relevant variables within this test.

On QNX, the Hailo PCIe driver runs as a user-space resource manager. The inference thread communicates through QNX message-passing primitives, and the interrupt completion path is priority driven. This gives the application a more predictable route from NPU completion back to the blocked inference thread.

On Linux, the same flow uses a kernel module and the generic interrupt path. PREEMPT_RT improves latency compared with stock Linux, but the system can still encounter workqueues, RCU activity, timer work, and threaded interrupt handling. These operations are often small individually, but in a one-frame-at-a-time inference loop they can accumulate into measurable run-to-run and frame-to-frame variation.

The DMA-buffer difference also matters. In the trace, Linux issued 21,441 VDMA_BUFFER_SYNC operations per 10,000-frame run. QNX issued none. Each individual sync is small, but the aggregate adds control-plane work and contributes to a wider latency distribution.

In addition, QNX processed the same workload with 35% fewer driver-level operations, reducing software overhead between the application and NPU. This allowed more system resources to be focused on inference processing rather than control-path management.

8. Implications for Physical AI Devices

Physical AI combines perception, planning, and action in devices that interact with the real world. In this context, the AI inference result is part of a larger timing chain. The system may also be handling sensor acquisition, control loops, actuator commands, diagnostics, logging, networking, and safety monitoring.

A faster average inference time helps, but repeatability is what makes the behavior usable. If a perception stack must deliver a decision within a defined budget, engineering teams need to know not only what the average frame does, but how much the frame timing moves from run to run and under sustained operation.

The benchmark shows that QNX provides a stronger measured basis for this timing argument within the tested configuration. The combination of higher throughput, lower mean latency, lower variance, fewer driver calls, and cooler operation gives customers more room to build predictable AI-enabled systems without moving the inference path out of the QNX environment.

For a customer already using QNX in a safety-relevant system, this is strategically important. It reduces the need to introduce a separate Linux environment solely for NPU acceleration and helps preserve a more unified software architecture for real-time, safety and lifecycle management.

9. Outcome

QNX Drives the Hailo-8 in Production-style Execution

QNX SDP 8.x ran HailoRT v4.14.0 on the Hailo-8 AI HAT+ and completed one million MobileNet v1 inferences across 100 consecutive benchmark runs without a failed inference. The PCIe driver, VDMA subsystem and C++ HailoRT API surface operated as expected in the tested configuration.

The Performance and Determinism Argument

Within this benchmark, QNX did more than match Linux. It measured ahead across the reported KPIs (Key Performance Indicators). The average results settle the throughput and latency question, while the variance results are the stronger evidence for determinism and production relevance.

The Safety and Validation Argument

For ISO 26262, DO-178C, IEC 62304 or similar safety and assurance contexts, timing evidence must be repeatable enough to support a credible argument. The QNX results provide a more stable measurement set for timing analysis than the Linux results in this benchmark, particularly because QNX shows 14× tighter run-to-run throughput variation and 2.6× tighter per-frame latency variation.

10. Limitations and Future Work

The benchmark is intentionally narrow. It uses a single hardware platform, one NPU, one model, and a synchronous single-frame workload. The results should not be generalized to all NPUs, all models, or all deployment patterns without additional testing.

Power was not measured because the Raspberry Pi 5 AI HAT+ configuration used for the test did not expose a suitable interface for precise board-level power measurement. As a result, this paper does not report energy per inference or sustained watts under load.

Future work should expand the benchmark across additional NPUs, larger Physical AI model families, active thermal-management scenarios and more production-like workloads such as streaming pipelines, multi-model perception stacks and closed-enclosure thermal profiles.

11. Conclusion

The benchmark holds hardware, model, dataset, compiler, runtime version, and cooling constant while changing only the operating system. Under those conditions, QNX delivered higher throughput, lower latency, and significantly tighter timing consistency than PREEMPT_RT Linux.

For conventional benchmarks, a 4.1% throughput gain is meaningful. For Physical AI devices, the more important result is predictability: 14× tighter run-to-run FPS variation and 2.6× tighter per-frame latency variation. Those differences can directly affect timing budgets, validation confidence, thermal headroom and the credibility of safety or certification arguments.

The conclusion is clear within the scope of this benchmark: when AI inference must support real-time physical systems, the operating system is part of the performance and assurance case. On this Hailo-8 MobileNet v1 workload, QNX provides the stronger measured foundation.

About QNX

QNX, a division of BlackBerry Limited, enhances the human experience and amplifies technology-driven industries, providing a trusted foundation for software-defined businesses to thrive. The business leads the way in delivering safe and secure operating systems, hypervisors, middleware, solutions, and development tools, along with support and services delivered by trusted embedded software experts. QNX® technology has been deployed in the world's most critical embedded systems, including more than 275 million vehicles on the road today. QNX® software is trusted across industries including automotive, medical devices, industrial controls, robotics, commercial vehicles, rail, and aerospace and defense. Founded in 1980, QNX is headquartered in Ottawa, Canada.

Learn more at qnx.com →