MLPerf Evaluation on Ryzen AI SoC

Noble Purpose United
Giulio Mantovi, Davide Paltrinieri

July 2, 2025

POLITECNICO
MILANO 1863

POLITECNICO MILANO 1863
NECST laboratory

Abstract

This study evaluates the performance and energy efficiency of AMD's Ryzen AI Neural Processing Unit (NPU) integrated within the System-on-Chip architecture for edge AI applications. Using the industry-standard MLPerf Inference benchmark, the research conducts a comprehensive comparative analysis of the NPU against the CPU and IGPU within the same SoC across standardized workload scenarios. The experimental results demonstrate superior computational performance and substantially enhanced energy efficiency of the NPU relative to other on-chip processing units across multiple benchmark scenarios. The analysis establishes the Ryzen AI NPU as an effective hardware accelerator with demonstrated adaptability to heterogeneous AI workloads, exhibiting optimal performance characteristics in both high-throughput and latency-constrained edge computing environments.

1 Introduction

1.1 Objectives

The primary goal of this project is to evaluate the performance and energy efficiency of the AMD Ryzen AI SoC NPU. To this end, we have defined the following key objectives:

1. Achieve a **valid MLPerf Submission**: Generate a valid and reproducible MLPerf Inference submission for the Ryzen AI platform to enable fair, standardized comparisons against other systems.

2. **Analyze energy efficiency**: Quantify the NPU's power consumption during inference to calculate its performance per watt, a critical metric for edge computing.

3. Evaluate **performance across MLPerf scenarios**: Assess NPU performance across key MLPerf scenarios (Offline, SingleStream, Server) to evaluate its suitability for different deployment contexts, from high-throughput to latency-sensitive tasks.

4. Compare **heterogeneous processing units**: Benchmark the NPU against the on-chip CPU and iGPU on the same workloads to understand their respective strengths and trade-offs.

1.2 Motivations

This research is driven by three key factors. First, the **emergence of Edge AI** demands powerful yet energy-efficient on-device hardware. The AMD Ryzen AI SoC, with its dedicated NPU, is a direct response to this trend and the focus of our analysis. Second, the spread of heterogeneous platforms necessitates a **standardized evaluation** for fair comparison. We adopt the industry-standard MLPerf benchmark to ensure our results are credible and reproducible. Finally, the **energy-efficiency challenge** is paramount for edge devices. Our work analyzes the NPU's performance per watt to understand the critical trade-offs between speed and power consumption.

2 Context

2.1 MLPerf Submission

MLPerf [4] was established by MLCommons on May 2, 2018, as a standardized benchmarking suite for measuring AI system performance. MLPerf provides benchmarks that enable reproducible performance comparisons across different machine learning hardware and software systems.

An **MLPerf submission** represents a formal entry that documents the performance characteristics of a machine learning system configuration. Each submission includes system specifications, performance measurements, accuracy validation, and compliance verification according to MLPerf rules.

The MLPerf benchmark suite evaluates machine learning performance across distinct categories. Training benchmarks assess model training speed to target accuracy, while Inference benchmarks measure inference performance on pre-trained models.

Deployment environments include *Edge* (resource-constrained), *Datacenter* (high-throughput servers), and *Mobile* (smartphones and tablets). Optimization strategies are split into *Open Division*, which allows unrestricted techniques meeting accuracy goals, and *Closed Division*, which enforces standardized implementations for fair comparison.

Benchmarks are also classified by maturity: *Research* benchmarks are experimental and support emerging workloads.

This work focuses on i*mage classification* with convolutional neural networks in the *Open Division* of *Inference* benchmarks within the *Research* category.

The inference execution scenarios define how workloads are presented to the system:

- *Offline Scenario* aggregates queries into batches and measures maximum throughput without latency constraints, measuring performance in samples per second.
- *Single Stream Scenario* sends queries sequentially, measuring end-to-end latency per query for edge computing environments and single-user applications.
- *Server Scenario* simulates production environments where queries arrive at variable rates and must be processed within specified latency bounds for real-time service deployment.
- *Multi-Stream Scenario* sends queries at fixed rates, measuring inference capacity per query batch while balancing throughput and latency requirements.

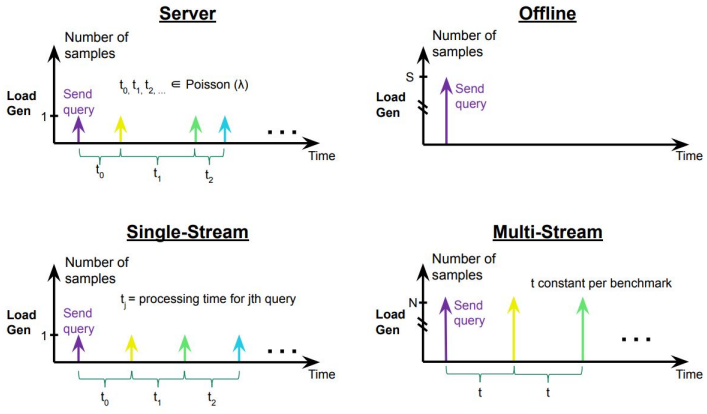

Figure 1: MLPerf Inference scenarios

2.2 Hardware and software

The AMD Ryzen AI Software [2] provides a comprehensive suite of tools and runtime libraries that enables deployment of pre-trained machine learning models on both the Neural Processing Unit (NPU) and integrated GPU (iGPU) through ONNX Runtime with Vitis AI Execution Provider integration. To optimize inference performance, models undergo quantization using AMD Quark Quantizer, which reduces numerical precision from FP32 to INT8 or BF16 formats. This process significantly decreases model size and accelerates inference throughput while maintaining acceptable accuracy levels. The underlying NPU hardware utilizes AMD's XDNA architecture

[5], featuring a 2D array of specialized tiles: Compute Tiles with VLIW cores and L1 memory, Memory Tiles providing shared L2 scratchpad, and Interface Tiles handling external communication. The architecture supports spatial partitioning into up to four independent partitions, enabling concurrent execution of multiple applications and maximizing hardware utilization for diverse ML inference workloads.

3 Implementation

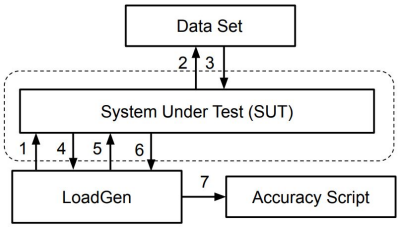

Figure 2: Architecture of MLPerf Inference benchmarking framework

Figure 2 illustrates the MLPerf Inference framework's core architecture and component interactions during evaluation. The **System Under Test (SUT)** executes machine learning inference, encompassing the complete hardware-software stack including accelerators (e.g., NPUs), inference frameworks (e.g., ONNX Runtime, PyTorch), optimization methods (e.g., batching, quantization), and application logic. The **LoadGen** generates inference queries, implements standardized benchmark scenarios, measures performance metrics, and produces reports. The **Query Sample Library (QSL)** manages LoadGen-dataset interaction through sample selection, memory management, and on-demand data access. This architecture enables standardized benchmarking across heterogeneous platforms and software stacks.

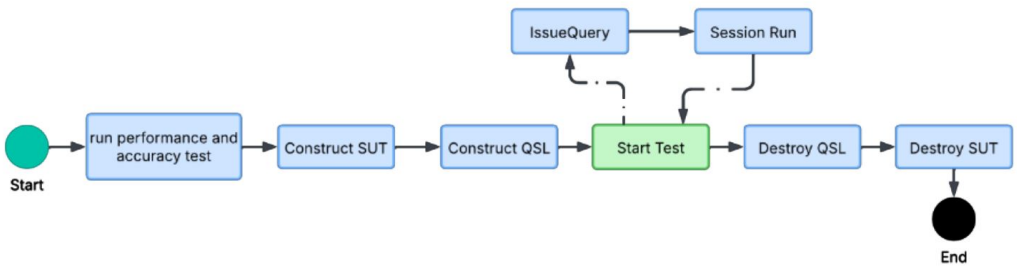

Figure 3: Code workflow - Offline and Single Stream scenario

This flow chart (figure 3) illustrates the MLPerf inference benchmarking workflow for both Offline and SingleStream scenarios due to their similarity.

The benchmarking process begins with `run_performance_test()` and `run_accuracy_test()`, which share identical execution flows but differ in MLPerf loadgen configuration parameters. The performance test targets throughput and latency metrics using `TestModePerformanceOnly`, while the accuracy test evaluates model prediction quality using `TestModeAccuracyOnly`. After SUT and QSL instantiation, the loadgen module's `StartTest()` function initiates benchmark execution with configured test settings. The loadgen framework then calls the registered `issue_queries()` callback to process inference requests. During execution, `issue_queries()` performs model inference using `session.run()` on query samples, then signals completion to the QSL via `QuerySamplesComplete()`. This creates a controlled feedback loop where loadgen manages query scheduling and timing while the callback handles inference computation.

The scenarios differ in their query distribution strategies. In SingleStream, loadgen invokes `issue_queries()` multiple times with individual samples to optimize for low-latency processing. Conversely, the Offline scenario typically calls `issue_queries()` once with all available samples, enabling batch processing for maximum throughput efficiency.

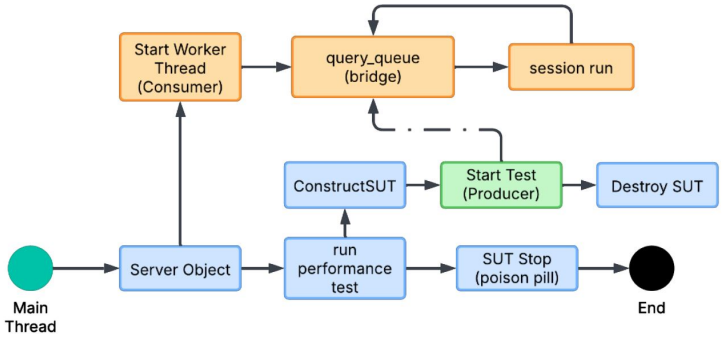

Figure 4: Code workflow - Server scenario

The flowchart 4 illustrates the MLPerf Server scenario implementation, which requires asynchronous multi-threaded architecture unlike the synchronous approach in Offline and SingleStream scenarios.

The Server implementation employs a producer-consumer pattern with dual threads. Upon initialization, a Server object spawns a worker thread (consumer) and establishes a thread-safe query queue bridging communication between the main thread (producer) and worker thread.

The benchmarking process follows standard initialization: `run_performance_test()`

constructs SUT and QSL components, then invokes `StartTest()`. When loadgen calls `issue_queries()`, instead of processing queries synchronously, it enqueues individual query samples into the shared queue without blocking.

The main thread continuously receives queries from loadgen and places them in the queue. Concurrently, the worker thread polls the queue, retrieving available queries, performing model inference using `session.run()`, and signaling completion via `QuerySamplesComplete()`.

This asynchronous design enables Server scenario requirements, maintaining consistent throughput while handling varying query arrival patterns. The queue buffers burst loads effectively.

Upon benchmark completion, standard SUT and QSL destruction occurs. Moreover, Server implementation requires additional cleanup through a poison pill mechanism that gracefully terminates the worker thread, ensuring clean thread termination.

3.1 Experimental setup

The experimental evaluation was conducted on two AMD Ryzen AI 9 HX 370 systems with integrated Radeon 890M Strix Point architecture, configured with AMD Ryzen AI software stack versions 1.3.1 and 1.4.0 respectively. The cloud-based system (version 1.4.0) operates through a JupyterLab environment.

The workflow 5 begins with ResNet50 model export to ONNX format, followed by *FP32* to *INT8* quantization via **AMD Quark**. Benchmark execution involves parallel MLPerf inference benchmarking and system monitoring through **XRT-SMI Tool** [3] for real-time NPU metrics and **AMDuProf Profiler** [1] for power profiling. The single-socket architecture enables accurate NPU power estimation through differential analysis of total system versus CPU-only power consumption.

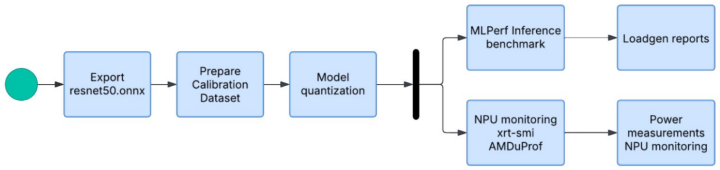

Figure 5: Experimental setup inference workflow

4 Results

This section details the power measurements and the performances obtained from the MLPerf evaluation of the AMD Ryzen AI SoC, comparing the CPU, iGPU and NPU in Offline, SingleStream and Server scenario.

4.1 Performance

A primary outcome of this work was obtaining a valid MlPerf submission which ensures that the following performance comparisons are standardized, reproducible and fair. We obtained the following results by averaging the performances of 5 runs for each processing unit.
In the Offline scenario, the key performance metric is throughput, measured in samples per second.

Table 1: Performance comparison for Offline (left) and SingleStream (right) scenarios.

	CPU	**GPU**	**NPU**
Throughput (samples/sec)	29,2	87,6	**176,2**
Mean latency per sample (ms)	34,488	11,366	5,566

	CPU	**GPU**	**NPU**
90th-percentile latency(ms)	33,14	11,9	**5,61**
QPS (queries/sec)	31,44	83,90	178,22

The evaluation revealed a substantial performance advantage for the NPU. The NPU is approximately 6x faster than the CPU and 2x faster than the GPU.
For the Single-Stream scenario, the critical metric is 90th-percentile latency and the results mirrored the Offline scenario results.

For testing the Server scenario we defined a target QPS (Queries Per Second) and a maximum latency constraint for each query.

Table 2: Server performances

	CPU	**GPU**	**NPU**
QPS (queries/sec)	31,19	49,06	49,18
Mean latency per query (ms)	2882,47	29,30	**15,18**

The CPU was unable to meet the performance target, resulting in an invalid run. Its processing speed was insufficient to handle the incoming query rate without violating the latency constraint due to extensive queue times. In contrast, both the GPU and NPU produced valid runs. Across multiple test configurations, the NPU consistently outperformed the GPU.

4.2 Power measurements

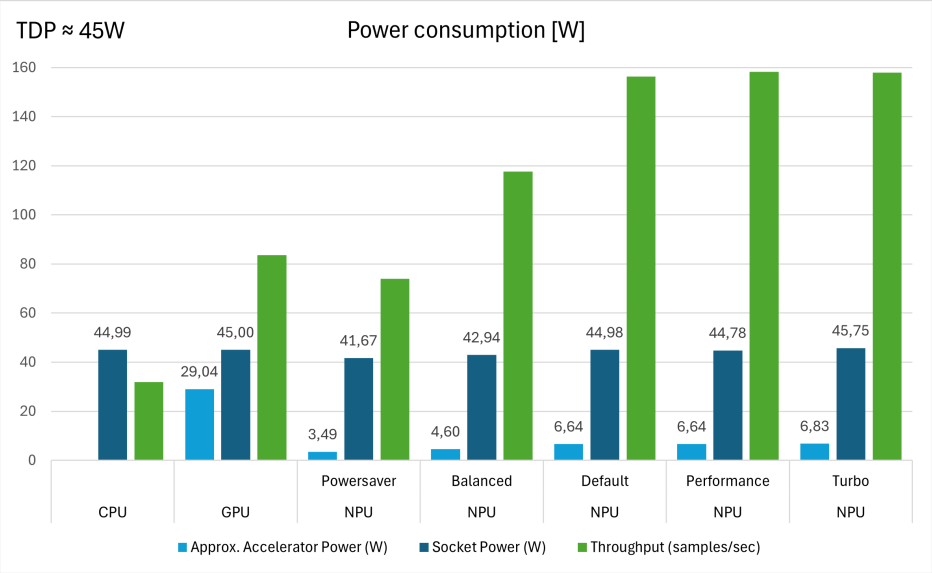

Figure 6: SoC power measurements

Power consumption was analyzed using **AMDuProf** to correlate inference throughput with SoC socket power. The NPU features several power modes, listed here from lowest to highest consumption: **Powersaver, Balanced, Default, Performance, and Turbo**.

Accelerator power was estimated as follows:

1. Measured total socket and CPU core power with AMDuProf.
2. Established a baseline non-accelerator power draw (∼2W) from a CPU-only run.
3. Calculated accelerator power by subtracting CPU and baseline power from the socket total.

At a system TDP of ∼45W, the NPU's efficiency was evident. For the same ∼45W power draw, the NPU (Default mode) delivered over 5x the CPU

throughput and 2x the GPU throughput. Furthermore, its Powersaver and Balanced modes offered high throughput while consuming significantly less power than the system's TDP.

5 Conclusions

This section summarizes the key findings and provides an interpretation of the results.

5.1 Performance

The MLPerf benchmark evaluation demonstrates consistent NPU superiority across all inference scenarios. In Offline and SingleStream scenarios, the NPU achieves 6× better performance than the CPU and 2× better than the GPU.

Scenario	**Metric**	**vs CPU**	**vs GPU**
Offline	Throughput	6× better	2× better
SingleStream	90th-percentile latency	6× better	2× better
Server	QPS (Target: 50)	190× better	–
	QPS (Target: 20)	5× better	2× better
	QPS (Target: 100)	–	58× better

Table 3: Relative performance in the three scenarios compared to CPU and GPU

The NPU's performance advantage is most significant in the Server scenario at high Target QPS. This gap occurs because Server latency includes queuing time (see Figure 4). At high query rates (50-100 QPS), the CPU and GPU become overwhelmed, causing long queuing delays that inflate latency from milliseconds to seconds. The NPU's superior performance reflects its higher sustainable throughput capacity of approximately 150 Target QPS.

5.2 Energy Efficiency

- **System-Level Performance per Watt**: When evaluating system-wide efficiency (throughput / socket power), Default mode proved to be optimal. Performance and Turbo modes were excluded from comparison as their performance was TDP-bottlenecked and equivalent to Default mode. The NPU is 5× more energy-efficient than the CPU and 2× more efficient than the GPU. Powersaver and Balanced modes achieve excellent performance-per-watt while consuming less overall power.

- **NPU-Specific Performance per Watt**: When considering only NPU power consumption, Balanced mode is most efficient, making it optimal for maximizing performance relative to direct power draw. Powersaver mode remains ideal for power-constrained environments.

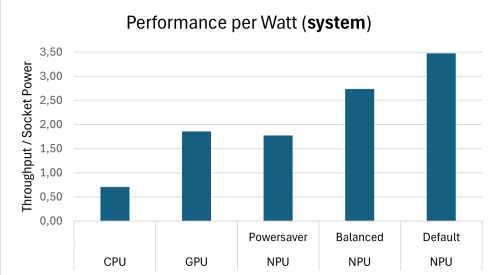

Figure 7: Performance per Watt (SoC Socket Power).

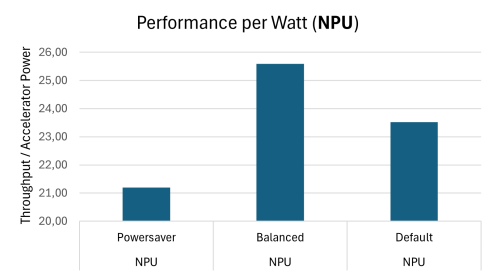

Figure 8: Performance per Watt (Estimated NPU Power).

5.3 Challenges

During our research we faced several challenges. The most significant was the lack of a direct NPU power interface, which forced us to estimate consumption. Furthermore, a system-wide TDP bottleneck blocked the evaluation of high-performance modes, and software incompatibilities prevented the MultiStream scenario from running and caused BF16 models to fall back to the CPU.

5.4 Main claim and future work

In summary, the main conclusion of this work is that the **Ryzen AI NPU outperforms the CPU and GPU across all tested ML scenarios, offering superior performance and performance per watt in both high-throughput and power-constrained environments.**
Next steps involve analyzing the performance and power trade-offs of **BF16 versus INT8 quantization**. We will also use the **AMD AI Analyzer** for a granular analysis of workload distribution across the heterogeneous processing units.

References

[1] AMD. *AMD uProf – Performance Analysis Tool*. https://www.amd.com/en/developer/uprof.html. 2024.

[2] AMD. *Ryzen AI Software*. https://www.amd.com/en/developer/resources/ryzen-ai-software.html. 2024.

[3] AMD. *XRT and SMI – Ryzen AI Software*. https://ryzenai.docs.amd.com/en/latest/xrt_smi.html. 2024.

[4] Vijay Janapa Reddi et al. *MLPerf Inference Benchmark*. 2019. arXiv: 1911.02549 [cs.LG].

[5] Alejandro Rico et al. "AMD XDNA NPU in Ryzen AI Processors". In: *IEEE Micro* 44.6 (2024), pp. 73–82. DOI: 10.1109/MM.2024.3423692.