A Power-Centric Methodology to Characterize Edge AI SoC with Limited Telemetry Capabilities

Giulio Mantovi*, Davide Paltrinieri*, Giuseppe Sorrentino, Christian Pilato, Davide Conficconi
Dipartimento di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano
{giulio.mantovi,davide.paltrinieri}@mail.polimi.it, {giuseppe.sorrentino, christian.pilato, davide.conficconi}@polimi.it

***Abstract*—The rapid growth of Edge AI is driving the adoption of on-device accelerator systems to deliver high throughput under a limited power budget, making energy efficiency a paramount constraint. Moreover, the increasing heterogeneity of System-on-Chip (SoC) motivates the need for standardized and reproducible AI evaluation methodologies for fair system comparison, such as the industry-standard Linux-centered MLPerf. This work focuses on AMD's Ryzen AI SoC as a representative system integrating a Neural Processing Unit (NPU), an integrated GPU (iGPU), and a CPU, with a Windows-based software stack. First, we bridge the MLPerf inference framework with the Ryzen AI platform by introducing a dedicated bridging layer. Using the MLPerf Inference benchmark, we conduct a comprehensive comparative analysis of NPU, iGPU, and CPU across standardized image classification workloads. However, the lack of, or limited availability of, fine-grained power telemetry hinders in-depth power and performance-per-watt analysis. Therefore, this work proposes a power-centric benchmarking methodology to provide a deeper understanding of the NPU power contributions to the SoC. Overall, the NPU consistently outperforms both the CPU and the iGPU in both performance and energy efficiency. When maximizing performance, the NPU achieves 12.2× and 2.1× speedups over CPU and iGPU, respectively. Instead, when targeting energy efficiency, the NPU delivers 44.3× and 6.8× improvements over CPU and iGPU, respectively. Finally, under tighter latency constraints, CPU–GPU throughput drops by 62.5 % when moving from a 100 ms to a 50 ms server latency constraint, whereas CPU–NPU degrades by only 13.9 %, enabling predictable latency-bound inference at the edge.**

***Index Terms*—MLPerf, Ryzen AI Software, NPU, Performance-Energy trade-off, Benchmarking**

I. Introduction

The growing interest in Edge AI is increasing the demand for on-device inference platforms that provide high throughput under tight power constraints [1]–[5]. This requirement, combined with real-time latency demands, has driven the adoption of heterogeneous System-on-Chip (SoC) architectures [6], [7].

In this context, AMD Ryzen AI SoCs provide a compelling heterogeneous platform by integrating a CPU, integrated GPU (iGPU), and Neural Processing Unit (NPU) within a single device [8], [9]. The NPU architecture (XDNA) shares structural similarities with Coarse-Grained Reconfigurable Arrays (CGRAs), featuring a spatially arranged array of compute and memory tiles with explicit dataflow control [10]. However, evaluating the suitability of each processing element for a given workload poses two key challenges. First, comparisons are often biased by software-stack heterogeneity, as the CPU, iGPU, and NPU are typically accessed through different runtimes and libraries, obscuring hardware-specific contributions [11]–[13]. Second, power and energy-efficiency analysis of emerging NPUs is hindered by the lack of, or limited availability of, fine-grained power telemetry, especially compared to more established backends such as GPUs. Indeed, prior studies emphasize latency and throughput, while power analysis of heterogeneous backends remains less standardized [14].

To address this need, MLPerf Inference [15] provides a standardized benchmarking methodology for reproducible evaluation in throughput- and latency-oriented inference scenarios. However, integrating its Linux-based infrastructure with the Windows-based Ryzen AI SW stack is non-trivial and the device is not officially supported.

To overcome these limitations, we propose a **power-centric, MLPerf-driven benchmarking** strategy for Ryzen AI SoCs[1]. We devise a bridging layer to enable the evaluation of Ryzen AI SoCs using MLPerf, ensuring a fair comparison of the three on-chip hardware layers – CPU, iGPU, and NPU – under identical experimental conditions. Using ONNX Runtime as a unified inference frontend, we target each backend by selecting its execution provider, keeping all software components identical across executions. Thus, we isolate backend-specific contributions in performance and energy efficiency. To summarize, our contributions are:

- A **bridging layer** to integrate Ryzen AI into the MLPerf suite. By leveraging ONNX, our workflow enables seamless evaluation of each SoC compute backend (§III-A).
- **MLPerf-compliant dataset and scenarios** to evaluate Ryzen AI SoC, comprising the Query Sample Library (QSL) and multiple custom Systems Under Test (SUTs) designed according to MLPerf guidelines (§III-B).
- A **power-centric profiling methodology** that enables energy-efficiency analysis despite limited NPU telemetry, by deriving accelerator-attributed power through a differential measurement approach based on system- and CPU-level power readings (§III-C).

Using an image classification workload across multiple MLPerf inference scenarios, we compare CPU, iGPU, and NPU execution in terms of throughput, latency, and energy efficiency. Results show the NPU as the most effective backend for edge inference, providing high performance while remaining robust under strict latency constraints. A performance-

*Both authors contributed equally to this research.

[1]https://github.com/necst/ryzenAI-MLPerf

optimized NPU configuration achieves up to 12.2× and 2.1× speedups over the CPU and iGPU, respectively, while an energy-optimized configuration improves efficiency by 44.3× and 6.8×. Under tighter latency constraints, CPU–iGPU throughput drops by 62.5 % when moving from a 100 ms to a 50 ms server target, whereas CPU–NPU decreases by only 13.9 %, making it better suited for latency-bound inference.

II. Background

AMD **Ryzen AI SoCs** integrate a CPU, an iGPU (also named RDNA), and a dedicated NPU on the same die, enabling heterogeneous execution under tight power constraints. The NPU, based on AMD's XDNA architecture, features a spatially reconfigurable array of compute and memory tiles [8], [16] driven by an explicit dataflow control [9]. Its structure resembles the one of a CGRA [10]. Each tile is a statically scheduled Very-Long Instruction Word (VLIW) unit supporting INT8 and BF16 efficient AI inference formats. The programmer must explicitly control tiles orchestration and their data movement. In this work, we target a Strix Point Ryzen AI SoC featuring the second-generation XDNA architecture (XDNA2), illustrated in Figure 1. Note that other generations mainly differ in the spatial array dimension.

We deploy inference using **Ryzen AI SW**, AMD's official framework built on ONNX Runtime [17], which integrates the providers to target CPU, iGPU, and NPU. Model optimization is supported via AMD Quark Quantizer [18], enabling reproducible exploration of performance–power trade-offs. This stack abstracts the complexity of fine-grained control and proposes automatic layer mappings on the target NPU.

MLPerf Inference [15] is an industry-standard benchmark developed by MLCommons to evaluate machine learning inference performance under reproducible conditions. It defines standardized workloads, datasets, accuracy targets, and compliance rules across four execution scenarios: *Offline* (throughput-oriented), *SingleStream* (single-query latency), *Server* (variable arrival rates with latency constraints), and *MultiStream* (fixed-rate query batches).

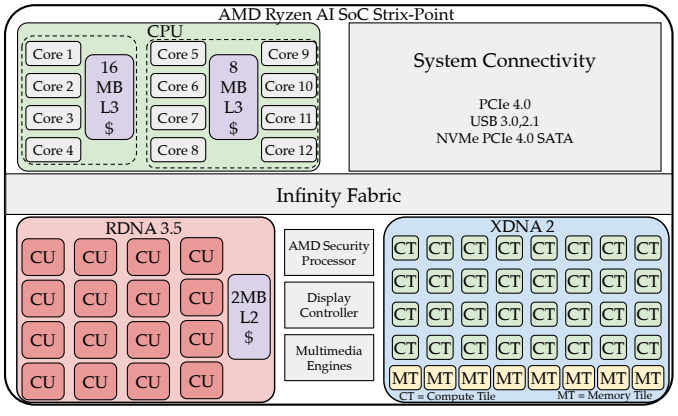


Fig. 1. Ryzen AI SoC Strix Point. The heterogeneous SoC integrates a 12-core CPU (organized into two clusters with 16 MB and 8 MB of cache), an RDNA 3.5 iGPU with 16 Compute Units (CUs), and an XDNA 2 NPU featuring eight columns, each coupling four compute tiles with a memory tile.

III. Implementation

To benchmark our architecture, we extend MLPerf by adapting its toolflow to the Ryzen AI software framework. §III-A introduces the *bridging layer* that maps MLPerf's Linux-based workflow to the Ryzen AI toolchain, enabling modular benchmarking pipelines for fair evaluation of each compute layer. §III-B describes the SUTs implemented to cover all MLPerf scenarios, ensuring compliant and comprehensive evaluation. Finally, §III-C presents our empirical strategy for power profiling in the absence of fine-grained telemetry.

A. Bridging MLPerf & Ryzen AI SoC

MLPerf Inference is widely adopted to ensure reproducible and comparable inference benchmarking across heterogeneous platforms. However, the MLPerf reference implementation does not support AMD Ryzen AI SoCs due to incompatibility with the device's software stack and operating system. Indeed, the device relies on Ryzen AI SW, which primarily runs on Windows-based devices, exposing heterogeneous backends (CPU, integrated GPU, and NPU) via ONNX Runtime execution providers. To enable MLPerf-compliant benchmarking on Ryzen AI without altering MLPerf semantics, we implement a dedicated *bridging layer* connecting MLPerf LoadGen to the Ryzen AI inference stack.

Figure 2 summarizes the proposed end-to-end workflow. From the user perspective, the pipeline follows the standard MLPerf interface: the target model, dataset, and execution scenario are specified, and query generation and metric collection are handled by the unmodified *LoadGen* component ①. To ensure Ryzen AI support in MLPerf, we introduce a dedicated *bridging layer* that adapts LoadGen requests to the Windows-based execution environment and orchestrates scenario execution on the Ryzen AI SoC ②. On top of this bridge, we implement MLPerf-compliant SUTs, responsible for scenario-specific scheduling and runtime interaction. During execution, the SUT interacts with an ad-hoc QSL, which retrieves input samples from main memory, applies required preprocessing, and prepares ready-to-use inputs while minimizing data movement ③,④. Inference is then executed through ONNX Runtime on the selected backend ⑤, targeting the *CPUExecutionProvider*, *DmlExecutionProvider* (iGPU), or *VitisAIExecutionProvider* (NPU).

All remaining factors (model, dataset, preprocessing, query generation, and scenario parameters) are kept identical to isolate the underlying hardware contribution. The final results are reported via standard MLPerf LoadGen outputs, enabling consistent, reproducible comparisons across backends.

B. Ryzen-based System-Under-Test

MLPerf scenarios impose different constraints on query generation and scheduling, making the SUT a key component for a fair comparison across target systems.

Upon device-MLPerf integration, we implement a set of scenario-specific SUTs that faithfully reproduce the query arrival and completion semantics defined by MLPerf. A key limitation of the current Ryzen AI NPU stack is the support

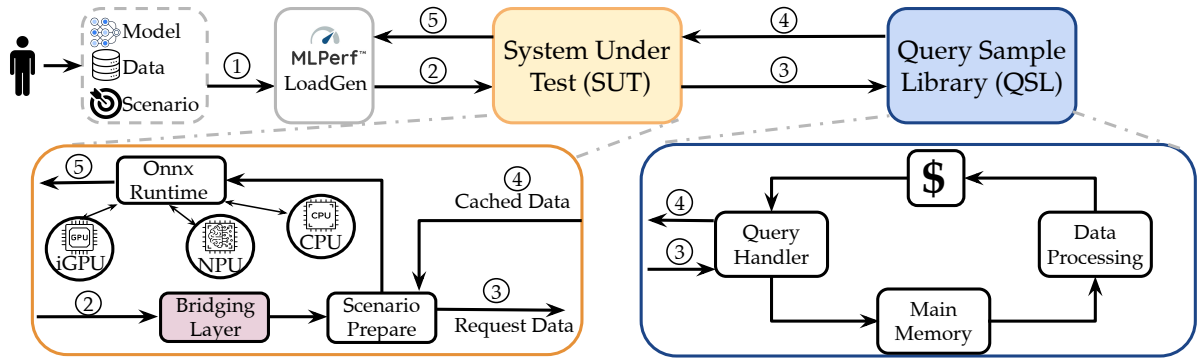


Fig. 2. MLPerf-based workflow for an AMD Ryzen AI SoC. The user provides the input dataset and the model, and the target scenario among those supported by MLPerf. The LoadGen collects the inputs and initializes the benchmarking pipeline. To integrate the Ryzen AI SoC into the workflow, we implement a custom SUT with a bridging layer that adapts MLPerf calls to the Ryzen AI stack. Each SUT fetches queries, generates scenario-dependent input streams, and executes the workload on the on-chip accelerator. The QSL is responsible for query management and data preparation, while ONNX Runtime provides a unified execution layer across the heterogeneous hardware components of the SoC.

for a fixed batch size of 1 [19], which directly impacts scenario coverage. Indeed, the *MultiStream* scenario requires concurrent execution of multiple samples per query, which is unfeasible given the NPU's fixed batch size of 1. Nevertheless, this batch size limitation is representative of real workloads like autoregressive LLM decoding, which inherently requires batch-1 [20]. Emulating MultiStream behavior would require serializing streams and executing them sequentially, effectively degenerating into a *SingleStream*-like scheduling model and leading to unfair comparisons with architectures that natively support multi-sample inference. For this reason, we evaluate only the *Offline*, *SingleStream*, and *Server* scenarios. However, the MLPerf inference guidelines also permit deriving MultiStream metrics from SingleStream, allowing us to complete our analysis.

In the following, we describe each scenario, highlighting the SUT behavior and the corresponding measurement objective.

Offline Scenario: LoadGen issues the entire query set to the SUT, and the system is required to maximize sustained *throughput*. Consistent with the execution constraints of the Ryzen AI SW framework for NPU, where inference effectively operates with a batch size of one [19], the Offline SUT adopts a fully sequential and deterministic execution model. As shown in Figure 3, once the whole workload is received, queries are processed in a strictly serialized order. For each query, the SUT synchronously retrieves the corresponding preprocessed sample from the QSL, executes single-sample inference through ONNX Runtime on the selected backend (CPU, iGPU, or NPU), and forwards the result to the results collector. The SUT repeats this procedure for all queries in the workload; then, the LoadGen aggregates the query results and returns the benchmark output to the user.

SingleStream Scenario: LoadGen issues one query at a time and enforces strictly serialized execution. The SUT processes each query in isolation and reports completion before accepting the next one; accordingly, the primary performance metric is the *per-query latency*. As illustrated in Figure 4, the SUT follows a fully synchronous execution model, processing one query at a time in strict sequence. Each query is received by the *Workload Receiver* and forwarded to the *Sequential Scheduler*, which enforces serialized execution. For each query, the scheduler synchronously retrieves the corresponding preprocessed input sample from the QSL, triggers single-sample inference through ONNX Runtime on the selected backend (CPU, iGPU, or NPU), and forwards the result to the *Results Collector*. Once completed, the LoadGen collects the result, and the workflow accepts another input query.

Server Scenario: Queries arrive asynchronously according to the arrival process enforced by LoadGen. The SUT

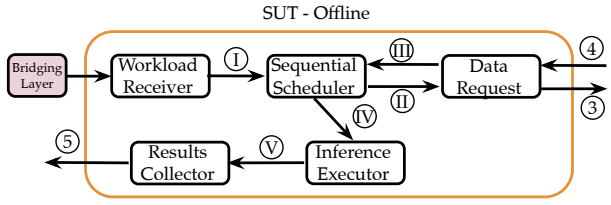


Fig. 3. Offline SUT workflow (I–II–3–4–III–IV–V–5). The workload taken from LoadGen is processed sequentially: for each query, the QSL provides the input, the backend runs the inference, and the LoadGen collects the final result, aggregating all queries at the end of the SUT.

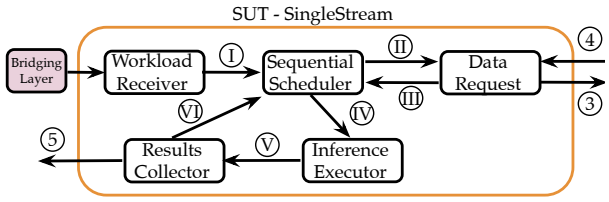


Fig. 4. SingleStream SUT workflow (I–II–3–4–III–IV–V–5–VI–IV...). The SUT processes one query at a time in a strictly synchronous way: once the query is received, the QSL provides data for performing a single-sample inference via ONNX Runtime. Then, the LoadGen collects and returns results before accepting the next query.

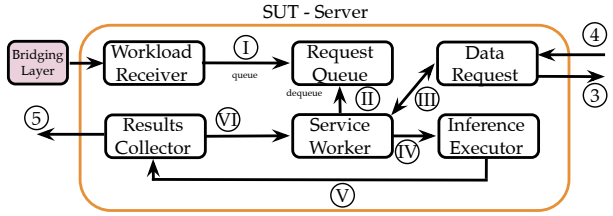


Fig. 5. Server SUT workflow (I–II–III–3–4–IV–V–5–VI–II...). Incoming queries are enqueued and served sequentially: for each request, the QSL provides the input, the device runs the single-sample inference, and the LoadGen collects the final result.

buffers incoming requests in a FIFO queue and serves them using a dedicated worker thread that repeatedly performs single-sample inference and reports completion. This scenario stresses both *tail latency* and *throughput under load*, as queue occupancy dynamically depends on the offered load and the service rate of the selected backend. Figure 5 describes the SUT, that operates as a continuously running service. Incoming queries are received by the *Workload Receiver* and enqueued into the *Request Queue*. A *Service Worker* repeatedly dequeues the next available request and processes it sequentially. For each query, the worker synchronously retrieves the corresponding preprocessed input sample from the QSL, executes single-sample inference through ONNX Runtime on the selected backend (CPU, iGPU, or NPU), and forwards the result to the *Results Collector*. Completion is reported back to LoadGen immediately, after which the worker resumes servicing the subsequent queued request.

Overall, the proposed SUTs faithfully adhere to MLPerf scenario semantics while reflecting the current execution constraints of the Ryzen AI platform. However, a unified MLPerf-compliant evaluation alone does not address discrepancies in power measurements across backends. Thus, we adopt a custom strategy to assess power costs fairly.

C. Power-centric Profiling Strategy

Due to the lack of backend-specific, fine-grained power telemetry (particularly for the NPU) we measure power costs using a system-level methodology based exclusively on AMDμProf. For each run, AMDμProf samples the **SoC socket power** (P_{sock}) and the **CPU package power** (P_{cpu}) using an identical configuration across all backends. Then, we compute the accelerator-side power (iGPU+NPU) as:

$$P_{\text{acc}} \triangleq P_{\text{sock}} - P_{\text{cpu}}. \tag{1}$$

Since AMDμProf does not expose separate GPU and NPU rails, we estimate accelerator power through a baseline subtraction approach. We first record an **idle baseline**, collected when the data is loaded into the cache but before inference execution, in accordance with MLPerf Power guidelines [21]. We denote idle averages by an overbar $\bar{(\cdot)}$ while computing run-time averages over the benchmark execution window. The incremental power attributed to the active accelerator will be:

$$\Delta P_{\text{acc}}^{\text{target}} \triangleq P_{\text{acc}} - \bar{P}_{\text{acc}}. \tag{2}$$

Although P_{acc} aggregates iGPU and NPU contributions, only one accelerator backend is active in each experiment. Thus, $\Delta P_{\text{acc}}^{\text{target}}$ is the accelerator-specific power cost at SoC level. Power consumption is calculated as the mean of per-run average power, while energy-efficiency metrics are derived by pairing these estimates with MLPerf performance results.

The Ryzen AI NPU further exposes multiple predefined operating modes (*Powersaver*, *Balanced*, *Performance*, and *Turbo* (with *Default* adapting to the Windows power mode) [22]), which trade performance for power by adjusting internal frequency and power limits. By using the Ryzen AI SW framework, we can configure such modes and, since the execution provider remains unchanged, we enable consistent evaluation of performance–energy trade-offs across them.

IV. Experimental Setup and Results

We performed our experiments on an AMD Ryzen AI 300 series SoC (Strix Point), through the AMD University Program Ryzen AI Cluster. We used Ryzen AI SW v1.6.1 to deploy the image classification model, paired with the corresponding `amdnpu` driver. We measured performance using the standard MLPerf Inference KPIs for each scenario, complemented by power consumption measurements collected via AMD μProf version 5.2.431.0. Power figures are estimated relative to an idle baseline following the methodology described in §III-C.

All results correspond to valid MLPerf Inference v5.0 submissions and comply with MLPerf performance and execution rules. Since MLPerf requires 10-minute benchmark runs, reported metrics are obtained by averaging 50 1-minute executions and validated against full 10-minute runs to ensure measurement consistency.

For the iGPU evaluation, we tested batch sizes ranging from 1 to 128 and selected the top-performing batch size (32). Additionally, we report batch size of 1, as it is the only supported value for the NPU [19].

§IV-A analyzes different deployment configurations in terms of latency. Ryzen AI SW enables heterogeneous execution by mapping workloads either to a single compute engine (CPU, iGPU, or NPU) or to combinations thereof. This analysis identifies the highest-performing configurations, independently of power consumption or energy efficiency considerations.

§IV-B evaluates power consumption, defined as the incremental power relative to the idle baseline, and energy efficiency on the same deployment configurations. This provides a comprehensive assessment of the target SoC for power-constrained and energy-sensitive deployment scenarios.

A. Latency Benchmarking Across Ryzen AI Configurations

We evaluated an image classification workload using the ResNet-50 v1.5 model quantized to INT8 precision. Following MLPerf specifications, inputs are drawn from the ImageNet validation set and preprocessed to 224×224 RGB images. By leveraging the proposed workflow and bridging layer, we integrate the Ryzen AI Software stack into MLPerf Inference, enabling valid MLPerf submissions.

TABLE I
PERFORMANCE COMPARISON ACROSS MLPERF SCENARIOS, NPU CONFIGURATIONS AND IGPU BATCH SIZES.

Scenario (latency constr. [ms])	Metric	CPU	iGPU (BS=1)	iGPU (BS=32)	NPU (Default)	NPU (Powersaver)	NPU (Balanced)	NPU (Performance)	NPU (Turbo)
Offline	Throughput*	27.5±1.0	75.8±0.4	158.0±1.0	266.2±1.6	122.7±0.7	218.1±1.6	269.9±1.1	**334.0±0.8**
SingleStream	90th-pct††	38.8±1.5	13.8±0.1	-	4.0±0.0	9.5±0.0	4.9±0.0	3.8±0.2	**3.1±0.1**
	99th-pct††	42.7±1.7	14.0±0.1	-	4.2±0.1	10.3±0.3	5.2±0.1	4.2±0.2	**3.4±0.0**
	QPS‡	28.2±1.0	75.9±0.7	-	261.9±3.3	119.7±0.7	220.7±4.4	300.6±37.6	**341.1±7.6**
MultiStream†	99th-pct††	341.9±13.2	111.8±0.8	-	33.8±0.4	82.4±2.6	41.8±0.7	33.8±1.3	**27.3±0.2**
Server 4 QPS (100)	QPS‡	3.8±0.0	3.8±0.0	3.8±0.0	3.8±0.0	4.0±0.0	4.0±0.0	4.0±0.0	4.0±0.0
	99th-pct††	61.2±19.7	41.5±2.1	66.1±3.2	20.5±0.3	28.5±1.4	21.6±0.3	20.6±0.2	**19.8±0.1**
Server 40 QPS (100)	QPS‡	25.4±6.5	39.3±0.0	39.3±0.0	39.3±0.0	39.3±0.0	39.3±0.0	39.3±0.0	39.3±0.0
	99th-pct††	> 100	66.9±1.4	82.0±2.4	22.2±0.3	38.8±0.5	24.3±0.2	22.2±0.4	**20.1±0.2**
Server 100 QPS (100)	QPS‡	-	77.3±0.1	97.4±0.0	97.5±0.0	97.5±0.0	97.5±0.0	97.5±0.0	97.5±0.0
	99th-pct††	> 100	> 100	> 100	25.7±0.4	93.9±4.5	31.3±0.2	26.0±0.4	**22.5±0.4**
Server 300 QPS (100)	QPS‡	-	-	-	257.9±3.4	120.3±0.6	205.4±1.9	257.0±4.8	299.6±0.0
	99th-pct††	> 100	> 100	> 100	> 100	> 100	> 100	> 100	> 100
Server Opt. QPS (100)	QPS‡	7.3±0.5	55.7±0.7	60.3±2.0	224.3±1.0	99.1±1.8	185.3±3.8	245.3±1.2	**322.2±2.1**
	99th-pct††	97.8±2.3	99.0±0.7	96.7±1.7	97.4±1.7	95.9±2.3	94.5±1.4	92.5±5.8	89.9±6.1
Server Opt. QPS (50)	QPS‡	-	20.9±1.4	-	193.2±2.9	71.0±2.4	160.3±2.6	208.1±2.8	**281.3±0.7**
	99th-pct††	> 50	49.4±0.5	> 50	47.8±1.8	49.4±0.4	46.8±2.6	47.9±2.6	47.7±2.6

* Throughput in samples/s. † MultiStream metrics are derived from SingleStream results following MLPerf inference guidelines. †† Latency percentiles in ms. ‡ QPS in queries/s.

Table I reports performance and latency across the supported Ryzen AI configurations and MLPerf scenarios. In the *Offline* scenario, MLPerf measures performance as throughput. In the *SingleStream* scenario, the benchmark suite measures latency at the 90th and 99th percentiles and reports performance as Queries Per Second (QPS). Finally, *Server* scenarios are characterized by sustained QPS and 99th percentile latency.

For the *Server* scenario, we evaluate fixed load rates of 4, 40, 100, and 300 QPS, as well as the MLPerf *Optimized* mode, in which LoadGen heuristically determines the maximum sustainable throughput. While LoadGen enforces the average query rate, individual arrivals follow a Poisson distribution, emulating realistic asynchronous workloads. According to MLPerf rules, *Server* runs must satisfy a 99th percentile latency constraint; otherwise, they are deemed invalid. Consistent with MLPerf inference guidelines [23], we derive MultiStream performance from SingleStream results as 8× the 99th percentile latency, an approach explicitly allowed for platforms without native concurrent scheduling support.

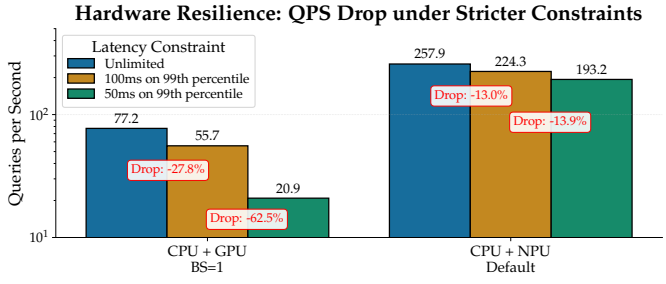


Fig. 6. Server scenario under different latency constraints for CPU–iGPU and CPU–NPU configurations. The iGPU-based setup is significantly more sensitive to tighter latency targets.

A key insight emerges when tightening *Server* latency constraints. As shown in Figure 6, reducing the target from 100 ms to 50 ms imposes a severe throughput penalty on the CPU–iGPU configuration (optimal BS=1 in this case), with sustainable throughput dropping by 62.5 %. In contrast, the CPU–NPU configuration remains highly resilient, sustaining 193.2 QPS under the 50 ms constraint – corresponding to only a 13.9 % reduction relative to its 100 ms performance, with similar trends observed across NPU power modes. These results demonstrate that CPU–NPU execution not only delivers higher performance but also provides the determinism required to meet strict service level agreements in edge-server deployments, a key challenge in ML inference systems [24].

B. Power and Energy Efficiency Analysis

We collected power costs using the AMDμProf *timechart* utility, following MLPerf Power guidelines [21]. By using a single tool, we avoid bias introduced by different measurement tools. We consider the *Offline* scenario to measure costs at maximum hardware utilization – multiple measurements are averaged over five independent 2-minute runs with a 100 ms sampling interval. This setup enables simultaneous monitoring of total socket power and CPU package power.

Figure 7 shows the throughput-power trade-off across the evaluated backends. The idle baseline power is 3.2 W. Offloading inference to the NPU substantially reduces total socket power: CPU-only and CPU–iGPU reach peak consumption of 48.3 W and 35.5 W, respectively, whereas CPU–NPU configurations operate within a 5.6 W to 16.9 W range, depending on the selected power mode.

NPU operating modes expose a clear performance–energy trade-off. The **Turbo** mode maximizes throughput, reaching

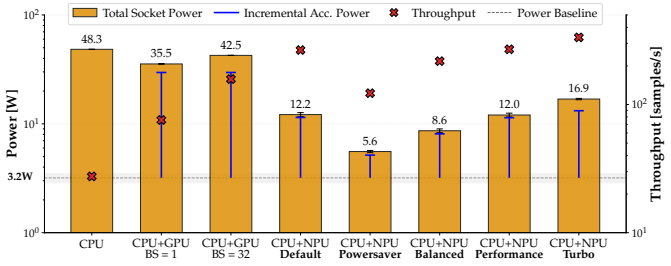


Fig. 7. Power consumption and throughput (log scale) across hardware configurations and NPU power modes.

334.0 samples/s at the cost of increased power consumption. In contrast, the **Powersaver** mode minimizes socket power to 5.6 W, corresponding to a 6.3× reduction compared to the CPU–iGPU configuration. Notably, even against the most efficient iGPU configuration (batch size 32), the **Powersaver** NPU mode delivers 122.7 samples/s at only 5.6 W, achieving a 7.6× power reduction for a 22 % throughput decrease.

To further quantify efficiency, we evaluate performance-per-watt at both system and accelerator levels.

At the system level, Figure 8 shows that the **Balanced** mode provides the optimal operating point, reaching 25.26 samples/J – approximately 44× higher efficiency than the CPU and 6.8× higher than the CPU–iGPU configuration. Default, Powersaver, and Performance modes exhibit comparable efficiency, while Turbo incurs a modest efficiency loss due to the nonlinear power cost of peak frequencies.

When isolating the accelerator contribution, Figure 9 shows that the **Powersaver** mode achieves the highest intrinsic efficiency, reaching 61.9 samples/J – nearly doubling higher-frequency modes. Such a result confirms that the NPU architecture is most efficient at lower voltage–frequency operating points. However, when accounting for full SoC power, fixed-platform overheads partially offset this advantage, highlighting the importance of evaluating heterogeneous systems using both system-level and component-level energy metrics [21].

V. Related Work

MLPerf Benchmarking: MLPerf provides a standardized framework for evaluating ML inference across heterogeneous systems, defining reference models, datasets, execution scenarios, accuracy targets, and compliance rules to ensure reproducibility and comparability [15], [25], [26]. MLPerf Power further standardizes power and energy measurements through unified guidelines on baselines, measurement boundaries, and reporting procedures [21], [27]. Together, these benchmarks enable fair comparisons across diverse accelerators and use cases, with increasing attention to energy efficiency [28]–[32].

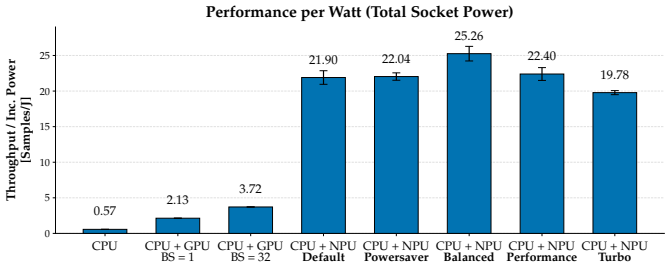


Fig. 8. Performance per Watt considering total SoC socket power.

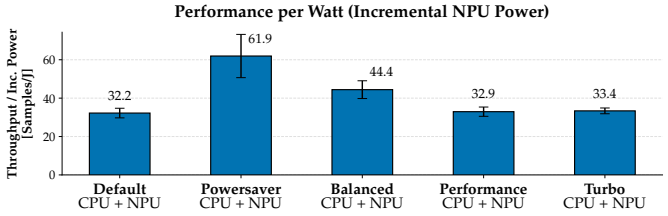


Fig. 9. Performance per Watt considering incremental NPU power only.

Ryzen AI Acceleration: Prior work has explored efficiency, programmability, and scheduling in heterogeneous AI systems, showing that workload-aware execution across CPU, GPU, and NPU can significantly reduce latency and energy consumption [14], [33]–[36]. In particular, [14] evaluates the performance of CNN and LLM workloads on AMD Ryzen AI platforms using Energy–Delay Product metrics. Beyond general edge settings, benchmarking efforts also target constrained environments [37], while architectural studies focus on the XDNA NPU design and integration [8], [16]. However, existing work does not provide standardized evaluation or scenario-driven inference benchmarks with systematic power attribution for Ryzen AI platforms, motivating our research.

VI. Conclusions

This work presented a benchmarking methodology to overcome the less standardized infrastructure of heterogeneous SoCs with NPUs. Thanks to the proposed approach and a custom MLPerf inference bridging layer, we could comprehensively evaluate the Ryzen AI heterogeneous SoC performance and energy efficiency. The results show that the NPU consistently offers the best balance among throughput, latency, and power consumption. In particular, CPU–NPU execution achieves superior latency determinism under strict Service Level Agreement constraints and significantly higher energy efficiency compared to both CPU-only and CPU–iGPU configurations. Moreover, the CPU–NPU setup maintains a better throughput-to-watt ratio and lower latency across all operating modes, making it the most balanced accelerator for low-latency edge inference on heterogeneous SoCs.

Acknowledgment

This work was supported by the European Commission in the context of the HORIZON EIC Cancerscan project (GA 101186829). This work was supported in part by the AMD University Program. The authors are grateful for the anonymous reviewers' feedback.

REFERENCES

[1] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[2] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7457–7469, 2020.

[3] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar, "Deepx: A software accelerator for low-power deep learning inference on mobile devices," in *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*. IEEE, 2016, pp. 1–12.

[4] R. Berzoini, E. D'Arnese, and D. Conficconi, "On how to push efficient medical semantic segmentation to the edge: the seneca approach," in *2022 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2022, pp. 104–111.

[5] R. Berzoini, E. D'Arnese, D. Conficconi, and M. D. Santambrogio, "Nerone: The fast way to efficiently execute your deep learning algorithm at the edge," *IEEE Journal of Biomedical and Health Informatics*, vol. 28, no. 3, pp. 1252–1260, 2023.

[6] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE transactions on wireless communications*, vol. 19, no. 1, pp. 447–457, 2019.

[7] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[8] Advanced Micro Devices, "Amd ryzen™ ai software architecture," Advanced Micro Devices, Tech. Rep., 2023.

[9] E. Hunhoff, J. Melber, K. Denolf, A. Bisca, S. Bayliss, S. Neuendorffer, J. Fifield, J. Lo, P. Vasireddy, P. James-Roxby *et al.*, "Efficiency, expressivity, and extensibility in a close-to-metal npu programming interface," in *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. IEEE, 2025, pp. 85–94.

[10] L. Liu, J. Zhu, Z. Li, Y. Lu, Y. Deng, J. Han, S. Yin, and S. Wei, "A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–39, 2019.

[11] Z. Jia, B. Tillman, M. Maggioni, and D. P. Scarpazza, "Dissecting the graphcore ipu architecture via microbenchmarking," *arXiv preprint arXiv:1912.03413*, 2019.

[12] J. Yao and M. Shepperd, "The impact of using biased performance metrics on software defect prediction research," *Information and Software Technology*, vol. 139, p. 106664, 2021.

[13] E. Soremekun, M. Papadakis, M. Cordy, and Y. Le Traon, "Software fairness: An analysis and survey," *ACM Computing Surveys*, vol. 58, no. 3, pp. 1–38, 2025.

[14] P. Kohli, R. Jayanth, N. Gupta, H. Fan, and V. Prasanna, "Performance-energy characterization of ml inference on heterogeneous edge ai platforms," in *2025 IEEE High Performance Extreme Computing Conference (HPEC)*, 2025, pp. 1–7.

[15] P. Mattson, H. Tang, G.-Y. Wei, C.-J. Wu, V. J. Reddi, C. Cheng, C. A. Coleman, G. F. Diamos, D. Kanter, P. Micikevicius, D. A. Patterson, and G. Schmuelling, "Mlperf: An industry standard benchmark suite for machine learning performance," *IEEE Micro*, vol. 40, pp. 8–16, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:214467842

[16] A. Rico, S. Pareek, J. Cabezas, D. Clarke, B. Ozgul, F. Barat, Y. Fu, S. Münz, D. Stuart, P. Schlangen *et al.*, "Amd xdna npu in ryzen ai processors," *IEEE Micro*, vol. 44, no. 6, pp. 73–82, 2024.

[17] Microsoft, "Onnx runtime documentation," https://onnxruntime.ai/.

[18] Advanced Micro Devices, "Amd quark documentation," https://quark.docs.amd.com/latest/intro.html.

[19] ——, "Ryzen ai software documentation – get started," https://ryzenai.docs.amd.com/en/latest/getstartex.html, accessed: 2026-02-06.

[20] A. Agrawal, N. Kedia, A. Panwar, J. Mohan, N. Kwatra, B. S. Gulavani, A. Tumanov, and R. Ramjee, "Efficient llm inference via chunked prefills," *SIGOPS Oper. Syst. Rev.*, vol. 59, no. 1, p. 9–16, Aug. 2025. [Online]. Available: https://doi.org/10.1145/3759441.3759444

[21] A. Tschand, A. T. R. Rajan, S. Idgunji, A. Ghosh, J. Holleman, C. Kiraly, P. Ambalkar, R. Borkar, R. Chukka *et al.*, "Mlperf power: Benchmarking the energy efficiency of machine learning systems from microwatts to megawatts for sustainable ai," in *2025 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. IEEE, 2025, also available as arXiv preprint arXiv:2410.12032.

[22] Advanced Micro Devices, Inc. (2026) Ryzen ai software documentation: xrt-smi. [Online]. Available: https://ryzenai.docs.amd.com/en/latest/xrt_smi.html

[23] MLCommons Association. (2026) Mlperf inference rules, version 5.1, section 4.1.1. [Online]. Available: https://github.com/mlcommons/inference_policies/blob/master/inference_rules.adoc

[24] M. Halpern, B. Boroujerdian, T. Mummert, E. Duesterwald, and V. Janapa Reddi, "One size does not fit all: Quantifying and exposing the accuracy-latency trade-off in machine learning cloud service apis via tolerance tiers," in *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Mar. 2019, p. 34–47. [Online]. Available: http://dx.doi.org/10.1109/ISPASS.2019.00012

[25] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schuemuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou *et al.*, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 446–459.

[26] MLCommons, "Mlperf inference benchmark v1.0," https://mlcommons.org/en/inference/, 2021.

[27] ——, "Mlperf power measurement specification," https://mlcommons.org/en/power/, 2023.

[28] K. Z. Ibrahim, T. Nguyen, H. A. Nam, W. Bhimji, S. Farrell, L. Oliker, M. Rowan, N. J. Wright, and S. Williams, "Architectural requirements for deep learning workloads in hpc environments," in *2021 International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2021, pp. 7–17.

[29] M. Emani, Z. Xie, S. Raskar, V. Sastry, W. Arnold, B. Wilson, R. Thakur, V. Vishwanath, Z. Liu, M. E. Papka, C. O. Bohorquez, R. Weisner, K. Li, Y. Sheng, Y. Du, J. Zhang, A. Tsyplikhin, G. Khaira, J. Fowers, R. Sivakumar, V. Godsoe, A. Macias, C. Tekur, and M. Boyd, "A comprehensive evaluation of novel ai accelerators for deep learning workloads," in *2022 IEEE/ACM International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2022, pp. 13–25.

[30] X. Sun, W. Cao, B. Crafton, K. Akarvardar, H. Mori, H. Fujiwara, H. Noguchi, Y.-D. Chih, M.-F. Chang, Y. Wang, and T.-Y. J. Chang, "Efficient processing of mlperf mobile workloads using digital compute-in-memory macros," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 4, pp. 1191–1205, 2024.

[31] G. Henry, P. Palangpour, M. Thomson, J. S. Gardner, B. Arden, J. Donahue, K. Houck, J. Johnson, K. O'Brien, S. Petersen *et al.*, "High-performance deep-learning coprocessor integrated into x86 soc with server-class cpus industrial product," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 2020, pp. 15–26.

[32] F. Ferdaus, X. Wu, V. Taylor, Z. Lan, S. Shanmugavelu, V. Vishwanath, and M. E. Papka, "Evaluating energy efficiency of ai accelerators using two mlperf benchmarks," in *2025 IEEE 25th International Symposium on Cluster, Cloud and Internet Computing (CCGrid)*, 2025, pp. 549–558.

[33] J. Saad-Falcon, A. Narayan, H. O. Akengin, J. W. Griffin, H. Shandilya, A. G. Lafuente, M. Goel, R. Joseph, S. Natarajan, E. K. Guha, S. Zhu, B. Athiwaratkun, J. Hennessy, A. Mirhoseini, and C. Ré, "Intelligence per watt: Measuring intelligence efficiency of local ai," 2025. [Online]. Available: https://arxiv.org/abs/2511.07885

[34] R. Jayanth, N. Gupta, and V. Prasanna, "Benchmarking edge ai platforms for high-performance ml inference," in *2024 IEEE High Performance Extreme Computing Conference (HPEC)*, 2024, pp. 1–7.

[35] 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*, vol. abs/2509.06794, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:281203150

[36] R. Karami, R. Patwari, H. Kwon, and A. Sirasao, "Exploring the dynamic scheduling space of real-time generative ai applications on emerging heterogeneous systems," 2025. [Online]. Available: https://arxiv.org/abs/2507.14715

[37] J. Wolf, M. Solé, I. Rodriguez, L. Kosmidis, and D. Steenari, "Design and implementation of an open source machine learning benchmarking suite for on-board space systems," *2025 European Data Handling & Data Processing Conference (EDHPC)*, pp. 1–7, 2025. [Online]. Available: https://api.semanticscholar.org/CorpusID:284922017