Benchmarking Deep Learning Convolutions on Energy-constrained CPUs

Enrique Galvez, Adrien Cassagne, Alix Munier, Manuel Bouyer

Benchmarking Deep Learning Convolutions on Energy-constrained CPUs

Enrique Galvez, Adrien Cassagne, Alix Munier, and Manuel Bouyer

Abstract. This work evaluates State-of-the-Art convolution algorithms for CPU-based CNN inference. Although most prior studies focus on GPUs or NPUs, CPU implementations remain comparatively under-optimized. Our first contribution is to provide fair benchmarking for embedded CPU inference. We evaluate direct, GEMM-based, and Winograd convolutions across modern CPUs from ARM®, Intel®, AMD®, and NVIDIA® vendors, considering both latency and energy efficiency. To the best of our knowledge, this is the first study to present a fair, cross-vendor comparison of CPU energy consumption using a high-resolution socket-level measurement platform. To validate our methodology, we further compare socket-level power measurements with estimates derived from model-specific registers (MSRs), finding that MSRs underestimate the power consumption of convolution inference by 10–30%. Our results show that the ARM® Cortex-A78AE CPU combined with an implicit GEMM convolution implementation offers the best trade-off between latency and power consumption, achieving ResNet50v1.5 inference in 102 ms with an average power of 25.3 W, corresponding to 2.58 J.

Keywords: Convolution algorithms · Benchmarking · Edge AI. · Energy-constrained CPUs · Energy measurements · Compute intensive

1 Introduction and Related work

Deep neural networks (DNNs) have become pervasive in modern embedded computer vision systems. Among them, convolutional neural networks (CNNs) remain the backbone of most classification [14] and detection pipelines [6], thanks to their relatively low computational and memory requirements as well as the maturity of available software frameworks [12,28]. Even with the rise of Transformer-based models such as DETR [2], CNN backbones still account for a significant portion of inference time: up to 30% in some configurations [18]. As a consequence, convolutions remain a central performance bottleneck in embedded inference of deep learning models.

The convolution operator is the key enabler of automatic feature extraction, but it also dominates the computational workload and energy demand during both training and inference [22]. In this paper, we show that convolutions account for more than 90% of the inference of popular networks such as ResNet50v1.5, VGG19 and GoogLeNet (see Section 4). This cost has motivated extensive research into optimized implementations for GPUs [17,23,25] and CPUs [5,7,21].

However, most of these studies target high-performance hardware or specific architectures, and little attention has been given to embedded CPUs, which are still widely deployed in systems where accelerators are absent or offloading is impractical due to small models or input sizes.

In this article, we address this gap by systematically benchmarking the inference of CNNs on CPUs. We consider CPUs commonly used in embedded and battery-powered systems from the main vendors: ARM®, Intel®, AMD® and NVIDIA®. Our study focuses on three complementary performance metrics: latency, power and energy consumption. These metrics are particularly critical in embedded contexts where energy efficiency and thermal constraints directly affect system design. This study extends the work of Dolz et al. [4], which was limited to ARM-based CPUs. Furthermore, we introduce a protocol leveraging a novel high-resolution power measurement system [3], enabling fine-grained characterization of the full system's energy profile during convolution execution. This work includes the following key contributions: (i) The characterization of a novel protocol based on high-frequency socket power measurement for benchmarking convolutions and the comparison of socket-measured power with power estimated using model-specific registers (MSRs); (ii) A fair and cross-vendor evaluation of State-of-the-Art convolution algorithms on modern embedded CPUs, supported by a multidimensional benchmark considering both inference latency and power consumption in order to guide algorithmic and hardware choices.

The remainder of this article is organized as follows. Section 2 presents the convolution algorithms evaluated in this study. Section 3 describes the hardware and software platforms considered. Section 4 outlines the experimental protocol. Section 5 compares socket-based power measurements with estimates obtained from MSRs. Section 6 analyzes the energy consumption of the convolution algorithms across the studied CPUs. Section 7 provides a benchmark of CNN inference across these CPUs. Finally, Section 8 concludes the article.

2 Convolution algorithms

The convolution operation consists of applying a filter with learnable values across an image. We consider 2-dimensional convolutions over 4-dimensional tensors, which are used by most CNNs in the context of computer vision. Moreover, since convolutions in such networks are rarely dilated or strided, we limit our study to undilated and unstrided convolutions.

Table 1. Notation for the main convolution parameters.

MB	batch size	IH, IW	input height, width
KH, KW	kernel height, width	OH, OW	output height, width
IC, OC	input, output channels	PH, PW	padding height, width

Using the notation described in Table 1, the dimensions of the input tensors can be written as: $MB \times IC \times IH \times IW$ and the dimensions of the weight tensors can be written as: $OC \times IC \times KH \times KW$. Following the same notations, the convolution operation can be described formally as:

$$dst[mb, oc, oh, ow] = bias[oc] + \sum_{ic=0}^{IC-1} \sum_{kh=0}^{KH-1} \sum_{kw=0}^{KW-1} src(mb, ic, ih, iw) \cdot weights[oc, ic, kh, kw], \quad (1)$$

where $ih := oh + kh - PH$ and $iw := ow + kw - PW$.

The output tensor *dst* is computed given an input tensor *src* and using learned values for *bias* and *weights* tensors. Each element of the output tensor is computed as a weighted sum of elements from the input tensor across the three dimensions of the kernel: channels (IC), kernel height (KH) and kernel width (KW). Figure 1 describes the convolution operator.

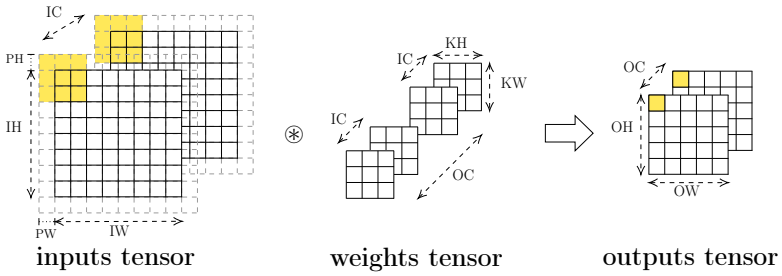

Fig. 1. Convolution Product.

In this article, we aim to minimize the latency of one CNN inference so we consider the batch size to be equal to 1. Predominant CNNs have convolution layers with 3×3 kernels ($KH = KW = 3$), 1×1 kernels and, less commonly, bigger kernels such that 7×7, for example. Later, we detail in Section 4 the temporal impact of different convolutions on the inference of common CNNs. The 1×1 convolutions can be implemented directly as a General Matrix Multiplication (GEMM), which is often preferred for performance. Other convolution kernels require specific implementations, discussed in the following paragraphs. Our study proposes a benchmark of State-of-the-Art's three most commonly used implementations of convolutions in modern deployment of CNN inference: direct, GEMM-based and winograd, described as follows.

Direct method [`direct`]. The convolution operator can be naively implemented using nested loops iterating through the output pixels and corresponding kernels. However, by reordering the loops and grouping the indices iterating over $[0, OC]$, $[0, IC]$, and $[0, OW]$ into fixed-size blocks, Zhang et al. [26] developed an optimized implementation of this computation, which we refer to as `direct`.

Explicit and Implicit Lowering [`im2row` and `gemm`]. These methods convert a convolution operation into a General Matrix Multiplication (GEMM)

using "lowering" techniques. The principle is to replicate data in order to transform input and weight tensors into matrices whose product corresponds to the convolution output. This approach can be particularly efficient, as GEMM is well optimized on modern CPUs [13] and GPU-based accelerators [1]. The lowering procedure is presented in Figure 2 and we refer to our explicit lowering convolution as `im2row` in the following. Although `im2row` involves a large overhead due to data movements, implicit lowering reduces this overhead by transforming small tiles of the tensor on-the-fly and computing the output result for each tile in parallel [17]. The default implementation of OneDNN [9] uses implicit lowering and is referred to as `gemm` in the following. These methods are particularly efficient on highly parallel architectures such as GPUs or TPUs [27].

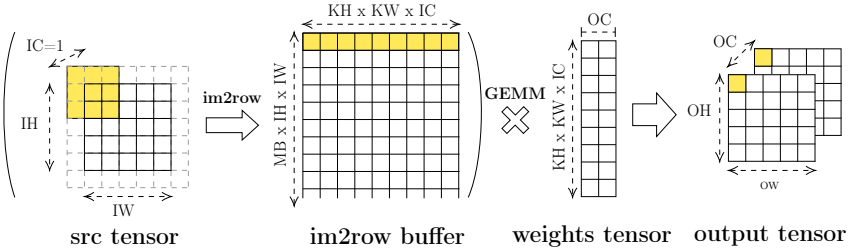

Fig. 2. Computing a convolution using `im2row`.

Winograd convolution [`wino`]. The last method is an extension of Winograd's algorithm [24] which reduces the number of floating-point multiplications of one-dimensional convolutions by taking advantage of redundancy patterns, at the cost of increasing the number of additions. To transpose this method to deep-learning convolutions, we followed the approach of Lavin and Gray [15], working on 3×3 convolution kernels. This algorithm is referred to as `wino` and its main benefit is to reduce the number of critical floating-point multiplications by 2.25 when the window size is 2. Such methods were implemented on GPUs [20], but are known to show floating-point imprecision with quantized data [19].

3 Experiments Platform

Our study targets CPUs typically deployed in embedded, battery-powered systems from the major processors vendors (ARM®, Intel®, AMD®, NVIDIA®). We selected the processors described in Table 2 for executing our workloads. The first class of CPUs considered in this study comprises the processors integrated into NVIDIA Jetson single-board computers, which are designed for computer vision. The CPUs of the AGX Orin and AGX Xavier boards both implement the ARM v8.2 instruction set: AGX Orin has a ARM Cortex-A78AE CPU and AGX Xavier has a NVIDIA Carmel CPU. A second class includes x86 processors. We evaluate a Ryzen 7 7840U based on the Zen 4 architecture and a Ryzen AI 9 HX 370 based on Zen 5. Finally, we benchmark an Intel Core Ultra 9 185H CPU from the Meteor Lake generation. Some of the studied architectures have heterogeneous CPUs, with performance cores, efficiency cores or low power efficiency

cores (respectively p, e and LPe cores). p-cores are the fastest CPU cores but they are more energy consuming than e and LPe cores that are slower. Some CPUs also allow 2-way simultaneous multithreading (2-SMT) on some of their cores. All platforms use Ubuntu distribution. The corresponding Linux kernels are reported in Table 2.

Table 2. Description of the targets with $\mathcal{T}$ the core type and $\mathcal{C}$ the cores number.

				CPU core					Software version	
Product & *Tag*	Vendor	TDP (Watts)	Release	$\mathcal{T}$	Architecture	$\mathcal{C}$	SMT	Proc. (nm)	Linux (Ubuntu)	GCC
Jetson AGX Xavier RAM 16 GB *(NV Carmel)*	NVIDIA	30 ≤	Oct'18	p	Carmel	8	1	12	4.09.201	11.4.0
Jetson AGX Orin RAM 64 GB *(ARM Cortex)*	ARM	60 ≤	Mar'23	p	Cortex-A78AE	12	1	8	5.10.120	11.4.0
Ryzen 7 7840U RAM 32 GB *(AMD Zen4)*	AMD	15–30	Jan'24	p	Zen 4	8	2	5	6.08.000	13.3.0
Core Ultra 9 185H RAM 32 GB *(Intel U9)*	Intel	35–115	Jul'24	LPe	Crestmont	2	1	5	6.14.000	14.2.0
				e	Crestmont	8	1	7		
				p	Redwood Cove	6	2	7		
Ryzen AI 9 HX 370 32 GB *(AMD Zen5)*	AMD	15–54	Oct'24	e	Zen 5c	8	2	4	6.08.000	13.3.0
				p	Zen 5	4	2	4		

The algorithms described in Section 2 are implemented into Intel's OneDNN framework version 3.4 [9]. This framework provides an abstraction level that allows one to implement deep learning primitives, with a built-in correctness and performance evaluation tool called BenchDNN. We compiled OneDNN with the `-fopenmp -O3` optimization flags. OpenMP runs threads to leverage tensor parallelism. The measurements have been first conducted using BenchDNN, focusing only on the convolutions. Later, a full inference across the ResNet50v1.5 network [8] is evaluated. Its execution relies on ONNX Runtime that links with our OneDNN implementations. ONNX Runtime v1.22.2 was used for all architectures except for NV Carmel which used version 1.17.0 for compatibility. It is worth mentioning that inter-operation parallelism is disabled. Only tensor parallelism inside OneDNN convolutions remains.

4 Experiments Protocol

The aim of this study is to benchmark latency and energy consumption of CPUs from several vendors for the inference of CNNs. An important thing to note in that we consider both convolution-level benchmark using BenchDNN and network-level inference benchmark using ONNX Runtime. In this context, Figure 3 compares the temporal cost of several convolutions over the inference of three popular CNN networks: ResNet50v1.5, VGG19 and GoogLeNet. The results have been obtained using ONNX Runtime built-in profiler on the AMD Ryzen7 7840U CPU. In order to measure per-layer timings, we set ONNX Runtime's `GraphOptimizationLevel` to `ORT_DISABLE_ALL` to disable layer fusion.

A first observation we can make from Figure 3 is that convolutions clearly dominate inference time of evaluated CNNs: often more than 90% of inference

time. Another observation is that 1×1 convolutions cost less than 3×3 convolutions individually but the high number of 1×1 convolutions in ResNet makes their cost non-negligible (42% of inference time). Despite the high cost of 1×1 convolutions in ResNet50v1.5, 3×3 convolutions cost approximately 52% of inference time. Another interesting observation is that predominant 3×3 convolutions have similar kernels between the networks. In particular, the most expensive convolutions of ResNet50v1.5 and VGG19 share the same dimensions. As a consequence to these observations, our convolution-level benchmarking will only focus on the most expensive 3×3 convolution in terms of computations: `MB1_IC256IH14_OC256OH14_KH3PH1` (which we refer to as `3x3_IC256_OC256` in Figure 3 and in the following). In the previous description format, each convolution parameter (from Table 1) is followed by its value. For instance, `MB1` means that minibatch is 1 corresponding to inference without batching.

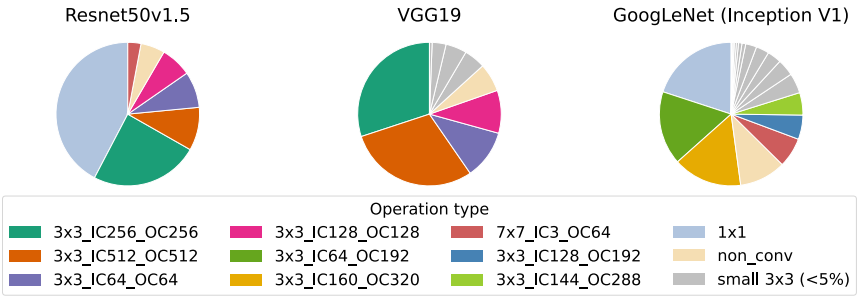

Fig. 3. Impact of relative latency of convolutions on CNN inference depending on their dimensions.

One of the contributions of this work is the experimental methodology. While its steps are not new on their own, the novelty of our approach resides in the use, for the first time, of high frequency measurement boards for monitoring socket power consumption of convolutions. The key methodology steps are:

1. Using `taskset` system command, we run the main program on a specific set of CPU HW threads (compact threads pinning on cores or HW threads).
2. We run a warm-up round of 200 convolutions or inferences, in order to ensure the stability of execution speed (stabilization of the CPU frequency).
3. We create a background thread that collects the energy measurements. It is pinning on a unused core when possible.
4. We run 1000 iterations of convolutions or inferences and gather the speed metrics returned by BenchDNN or ONNX Runtime
5. We stop the background thread and compute statistics over the energy measurements.

We used this methodology for all the following measurements, to ensure their pertinence and reproducibility. Section 5 analyzes the pertinence of socket measurements and provide important insights over the measurements uncertainty that can be expected from experimental results.

5 MSRs versus Socket Measure

To motivate our choice of high-frequency socket measurements over MSRs for power evaluation, we begin by comparing both methods. A first difference resides in the way that power is estimated in both cases. In one hand, MSR power is estimated by reading a register containing the energy consumption of the exposed power domain. By reading these registers at a certain frequency, the average power consumption is estimated for each interval. On the other hand, our high-frequency measurement system measures energy consumption between the power supply and the socket at 1000 samples/s for x86 CPUs [3] or 5000 samples/s for Jetson boards.

In order to evaluate the differences between MSRs power estimates and socket power measurements in the context of convolutions, we performed rounds of experiments for the `3x3_IC256_OC256` convolution, following the protocol described in Section 4. Since NVIDIA provides `jetson-power-tools` to precisely report power consumption across the components of Jetson boards, we used it to gather MSR values at 10 samples/s. For x86 CPUs, MSR power is gathered using `perf stat` to read the RAPL `/power/energy-pkg` event at 1 sample/s. We chose these sampling rates to minimize errors due to measurement overhead and the lack of energy updates [11].

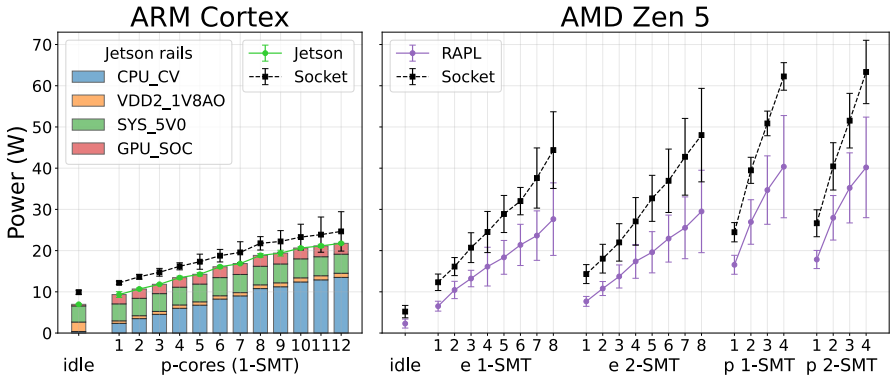

Fig. 4. MSRs versus socket measure of average power and dispersion (standard deviation) depending on multithreading.

Figure 4 provides a comparison between MSRs-based and socket-based power measurements for the inference of the `3x3_IC256_OC256` convolution for ARM Cortex A78AE (Jetson Orin) and AMD Ryzen AI 9 HX370 (x86) CPUs. A first observation we can make is that in both cases, measured MSR power is lower than socket power. However, the difference between socket power and Jetson power does not depend on multithreading while the difference between RAPL and socket power increases with multithreading. A possible explanation for RAPL's behavior is the fact that RAM is not part of the exposed power domain. As RAM usage increases with parallelism, this may explain the increasing difference

between RAPL and socket measurements for AMD Zen5. In addition to RAM, RAPL does not capture power supply losses, fans, storage or network, which may result in a non-constant overhead [10]. A last observation is that standard deviation of power measurements is high for both MSRs and socket readings.

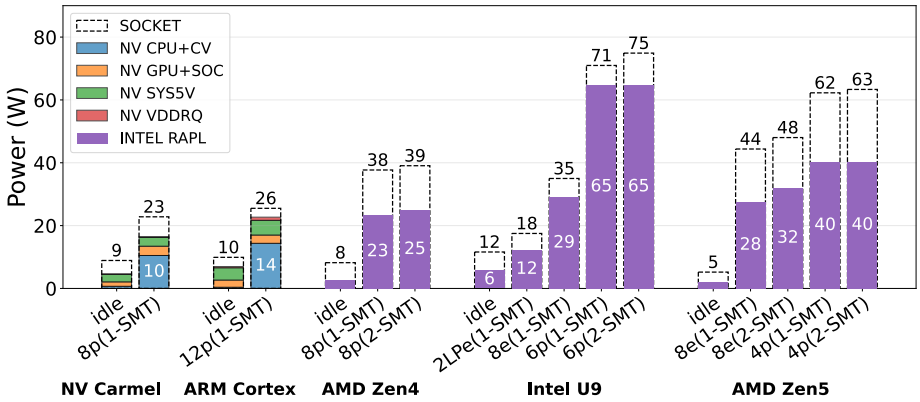

Fig. 5. Energy consumption measurements: MSRs and socket.

Using the same protocol, we extended the results displayed in Figure 4 to all studied architectures in Figure 5 the difference between MSRs and socket power measurements. Based on these experiments, we conclude that MSRs-based power measurements do not fully capture the total power consumption measured at the socket. In idle state, MSRs readings are more than 50% lower than the corresponding socket measurements, while during convolution computations, this discrepancy ranges between 10% and 30%. Another observation is that 2-SMT have a small and positive impact over average power consumption.

One may notice that `jetson-power-tools` provides the closest approximation of the actual power consumption (compared to RAPL). Despite the high dispersion of power measurements for both MSRs and socket-based readings, the mean power is more consistent across repeated socket measurements than across repeated MSRs measurements. Consequently, for the following experiments of this article, we rely exclusively on socket measurements to accurately benchmark the power consumption of each system.

6 Convolution performance depending on algorithm

Figure 6 presents the energy consumed to compute the `3x3_IC256_OC256` convolution depending on the number of CPU cores. Additionally, Figure 6 also compares the different algorithms. An important remark is that uncertainty propagated to energy is way smaller than uncertainty over power, due to the low variance of latency measurements.

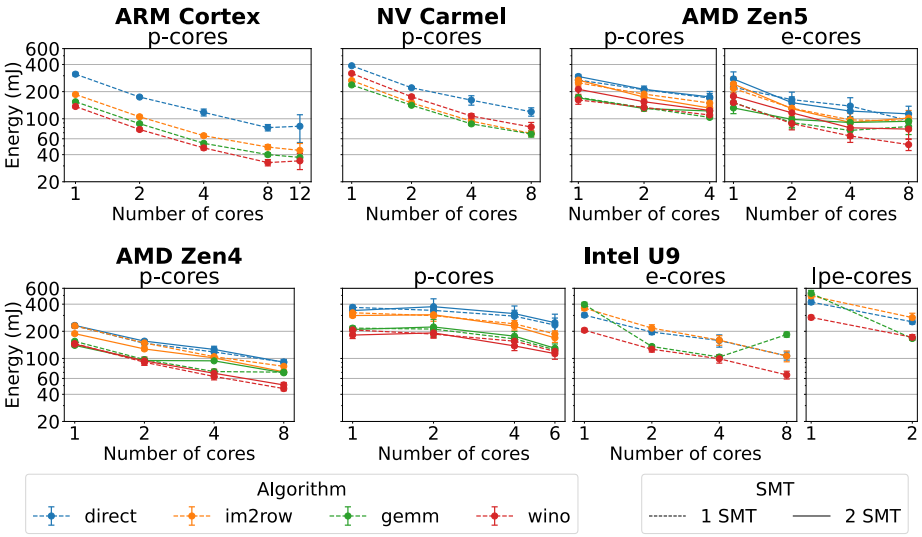

Fig. 6. Energy consumption of `3x3_IC256_OC256` convolution depending on algorithm and multithreading.

First, we observe that increasing the number of physical cores almost always decrease the energy consumption of the convolution. The reason is that increasing the number of cores further decreases latency than it increases power consumption. We also observe that 2-SMT has no significant impact on the average energy consumption of an operation. The reason is that convolution kernels already saturate the processor's backend execution units. SMT, which mainly helps when backend resources are underutilized, does not provide additional throughput in this context. A surprising result is that e-cores and LPe-cores consumes more than p-cores for the heterogeneous architectures. The reason is that low-power processors are much slower than they are energy efficient. Regarding the algorithms, `wino` and `gemm` are the most energy efficient because they are the fastest on almost every architecture. The main result of this benchmark is that the best algorithm-architecture configurations use 8 p-cores of the ARM Cortex A78AE or the AMD Zen 4, with the `wino` algorithm. Both configurations reach a power consumption that is less than 60 mJ.

7 Cross-vendor CPU benchmark for CNN inference

Results shown in Figure 7 are measured through a full inference of ResNet50v1.5. Since we target energy constrained devices, power consumption may be a discriminant criteria for the architectures, as well as inference latency. All the inference runs are executed on the same 640×640 image from COCO dataset [16] and we ensured the correctness of the predictions.

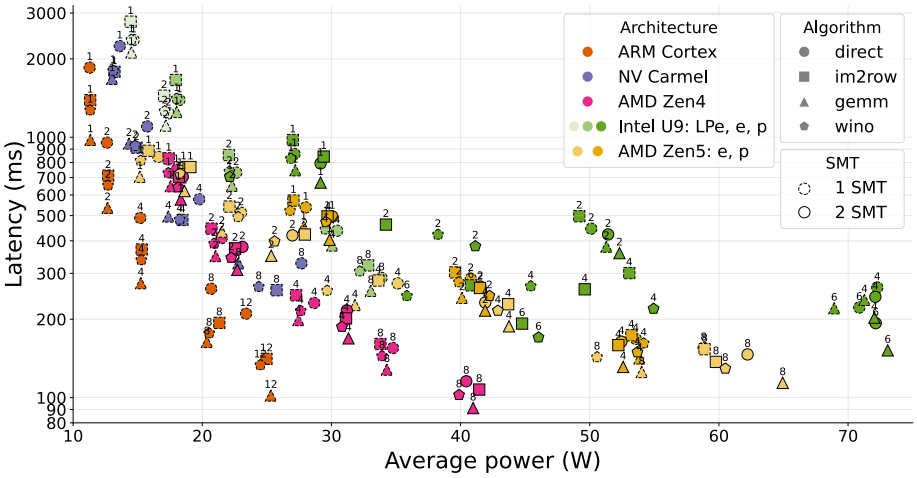

Fig. 7. Latency and power consumption depending on architecture, multithreading and algorithm for a full inference of ResNet50v1.5 .

A first observation is that focusing on the computational part of the convolution allows `wino` to take advantage of its arithmetic complexity optimization (see Figure 6) while in a full inference, significant data management acts in favour of implicit GEMM (`gemm`) implementation. In addition to that, Figure 7 clearly shows the existence of a trade-off between latency and average power consumption in our context. Moreover, among the studied architecture, ARM Cortex A78AE is the architecture offering the best trade-off between inference latency and average power consumption. If power budget is higher, AMD Zen4 can allow a faster inference. Additionally, NVIDIA Carmel CPU allow slow inference with a low power budget while, on the contrary, AMD Zen5 offers good latency at the cost of high power consumption. However, AtomMan X7 Ti offers sub-optimal performances on each one of its CPU coretypes.

8 Conclusion

This work introduces an original methodology based on high-resolution socket-level power measurements to benchmark CNN inference on CPUs commonly used in embedded and battery-powered systems. By extending prior studies to the major CPU vendors ARM®, Intel®, AMD®, NVIDIA® and leveraging precise socket-level energy instrumentation, we provide accurate insights into the real energy consumption of such CPUs. Our results demonstrate the relevance of socket-level energy measurements, which we prove to be more reliable and precise than MSR-based estimations. Our experiments also reveal a clear trade-off between inference latency and power consumption across all evaluated architectures. Overall, our benchmarks show that the ARM® Cortex-A78AE CPU combined with an implicit GEMM convolution implementation offers the

best trade-off between latency and power consumption, achieving ResNet50v1.5 inference in 102 ms with an average power of 25.3 W, corresponding to 2.58 J. Future work will expand this analysis to architectures with accelerators such as GPUs or NPUs, and further investigate the discrepancy between MSRs and socket energy measurements.

References

1. Abdelfattah, A., Haidar, A., Tomov, S., Dongarra, J.: Performance, design, and autotuning of batched gemm for gpus. In: Kunkel, J.M., Balaji, P., Dongarra, J. (eds.) High Performance Computing. pp. 21–38. Springer International Publishing, Cham (2016)
2. Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., Zagoruyko, S.: End-to-end object detection with transformers. In: Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I. p. 213–229. Springer-Verlag (2020)
3. Cassagne, A., Amiot, N., Bouyer, M.: Dalek: An unconventional and energy-aware heterogeneous cluster (2025), https://arxiv.org/abs/2508.10481
4. Dolz, M.F., Barrachina, S., Martínez, H., Castelló, A., Maciá, A., Fabregat, G., Tomás, A.E.: Performance–energy trade-offs of deep learning convolution algorithms on arm processors. The Journal of Supercomputing **79**(9), 9819–9836 (Jan 2023)
5. Dolz, M.F., Martinez, H., Alonso, P., Quintana-Orti, E.S.: Convolution operators for deep learning inference on the fujitsu a64fx processor. In: 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD) (2022)
6. Du, J.: Understanding of object detection based on cnn family and yolo. Journal of Physics: Conference Series **1004**(1) (apr 2018)
7. Georganas, E., Avancha, S., Banerjee, K., Kalamkar, D., Henry, G., Pabst, H., Heinecke, A.: Anatomy of high-performance deep learning convolutions on simd architectures (Aug 2018). https://doi.org/10.48550/ARXIV.1808.05567
8. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 770–778 (2016)
9. Intel: Oneapi deep neural network library, https://oneapi-src.github.io/oneDNN/
10. Jay, M., Ostapenco, V., Lefevre, L., Trystram, D., Orgerie, A.C., Fichel, B.: An experimental comparison of software-based power meters: focus on cpu and gpu (2023). https://doi.org/10.1109/CCGrid57682.2023.00020
11. Khan, K., Hirki, M., Niemi, T., Nurminen, J., Ou, Z.: Rapl in action: Experiences in using rapl for power measurements. ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS) **3** (01 2018). https://doi.org/10.1145/3177754
12. Khanam, R., Hussain, M.: Yolov11: An overview of the key architectural enhancements (2024), https://arxiv.org/abs/2410.17725
13. Kågström, B., Ling, P., van Loan, C.: Gemm-based level 3 blas: high-performance model implementations and performance evaluation benchmark. ACM Trans. Math. Softw. **24**(3), 268–302 (1998)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Commun. ACM **60**(6), 84–90 (May 2017)

15. Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks (Sep 2015). https://doi.org/10.48550/ARXIV.1509.09308
16. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) Computer Vision – ECCV 2014. pp. 740–755. Springer International Publishing, Cham (2014)
17. Lym, S., Lee, D., O'Connor, M., Chatterjee, N., Erez, M.: DeLTA: GPU Performance Model for Deep Learning Applications with In-depth Memory System Traffic Analysis. In: 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). pp. 293–303 (2019)
18. Ma, Y., Liang, W., Chen, B., Hao, Y., Hou, B., Yue, X., Zhang, C., Yuan, Y.: Revisiting detr pre-training for object detection (2023), https://arxiv.org/abs/2308.01300
19. Meng, L., Brothers, J.: Efficient winograd convolution via integer arithmetic (Jan 2019). https://doi.org/10.48550/ARXIV.1901.01965
20. Park, H., Kim, D., Ahn, J., Yoo, S.: Zero and data reuse-aware fast convolution for deep neural networks on gpu. In: Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis. CODES '16, Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2968456.2968476, https://doi.org/10.1145/2968456.2968476
21. Santana, A.d.L., Armejach, A., Casas, M.: Efficient direct convolution using long simd instructions. In: Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming. PPoPP '23, ACM (2023)
22. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.: Efficient Processing of Deep Neural Networks: A Tutorial and Survey. Tech. rep. (Aug 2017). https://doi.org/10.48550/arXiv.1703.09039
23. Wei, H., Liu, E., Zhao, Y., Yu, H.: Efficient non-fused winograd on gpus. In: Advances in Computer Graphics. pp. 411–418. Springer International Publishing, Cham (2020)
24. Winograd, S.: Arithmetic Complexity of Computations. Society for Industrial and Applied Mathematics (Jan 1980). https://doi.org/10.1137/1.9781611970364
25. Yan, D., Wang, W., Chu, X.: Optimizing batched winograd convolution on gpus. In: Proceedings of the 25th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. p. 32–44. New York, NY, USA (2020)
26. Zhang, J., Franchetti, F., Low, T.M.: High performance zero-memory overhead direct convolutions (Sep 2018). https://doi.org/10.48550/ARXIV.1809.10170
27. Zhou, Y., Yang, M., Guo, C., Leng, J., Liang, Y., Chen, Q., Guo, M., Zhu, Y.: Characterizing and demystifying the implicit convolution algorithm on commercial matrix-multiplication accelerators (Oct 2021). https://doi.org/10.48550/ARXIV.2110.03901
28. Zhuo, S., Bai, H., Jiang, L., Zhou, X., Duan, X., Ma, Y., Zhou, Z.: Scl-yolov11: A lightweight object detection network for low-illumination environments. IEEE Access **13**, 47653–47662 (2025)