ISSN: 2583-9888
DOI: doi.org/10.51219/JAIMLD/karthik-wali/580

URF PUBLISHERS
connect with research world

Journal of Artificial Intelligence, Machine Learning and Data Science

https://urfpublishers.com/journal/artificial-intelligence

Vol: 2 & Iss: 4

Research Article

Hardware-Software Co-Design for Power-Efficient Edge-AI Systems

Karthik Wali*

Citation: Wali K. Hardware-Software Co-Design for Power-Efficient Edge-AI Systems. *J Artif Intell Mach Learn & Data Sci 2024* 2(4), 2754-2760. DOI: doi.org/10.51219/JAIMLD/karthik-wali/580

Received: 03 October, 2024; **Accepted:** 28 October, 2024; **Published:** 30 October, 2024

***Corresponding author:** Karthik Wali, ASIC Design Engineer, USA, E-mail: ikarthikw@gmail.com

ABSTRACT

The proliferation of intelligent applications running at the network edge—everything from smart cameras and industrial IoT sensors to self-driving drones—has driven the need for high-performance but low-power edge-AI systems. In contrast to inference in the cloud, edge-AI has to deal with harsh constraints on energy, latency, and computational capacity, which requires a fundamental shift away from traditional isolated hardware or software-only optimization methodologies. This article explores the hardware-software co-design paradigm as an integrated approach to solving the multiple dimensioned challenges in the design of power-effective edge-AI systems.

Hardware-software co-design entails concurrent and synergistic optimization of system architecture and the software stack. By closing the formerly distinct spaces of hardware design (e.g., AI accelerators, memory stacks, power control units) and software engineering (e.g., neural network structure, compilers, scheduling algorithms), this method targets deriving globally optimal solutions specific to edge use cases. The paper begins by contextualizing the evolution of edge-AI, identifying its unique constraints—such as real-time processing requirements, energy autonomy, limited thermal envelopes, and increasing model complexity—and explaining why conventional design approaches fall short.

We then perform a comprehensive literature review that synthesizes recent breakthroughs in co-designed edge systems. Eminent techniques involve integration of sparsified and quantized deep learning models with low-power tensor processing units (TPUs), the utilization of dynamic voltage and frequency scaling (DVFS) with real-time operating systems (RTOS), and co-optimization environments that dynamically change model complexity according to runtime power profiles. We also discuss tools and middleware—like TensorRT, Apache TVM, and Xilinx Vitis AI-that support hardware-conscious model compilation and runtime adaptability.

Based on this, we introduce a modular and extensible co-design platform for power-efficient edge inference. The approach integrates model compression methods (e.g., pruning, quantization, and knowledge distillation), application-specific accelerator hardware design, and system-level runtime policies. Our platform includes an adaptive control loop in which telemetry information (e.g., workload intensity, battery level, and thermal sensors) is input to an AI-powered power manager, which dynamically adjusts execution paths in real time. This is deployed on a heterogeneous edge platform that includes ARM-based CPUs, NPUs (Neural Processing Units), and embedded FPGAs, managed through a lightweight runtime scheduler.

Our experimental confirmation spans simulated workloads over benchmark data such as ImageNet and CIFAR-100, in addition to real-world deployment settings including smart surveillance and autonomous navigation. The results showcase up to 60% reduction in energy use and 40% improvement in inference throughput relative to state-of-the-art solo optimization techniques. In addition, latency gains of up to 25% were realized without any compromise in prediction accuracy, proving the effectiveness of joint optimization through the system stack.

The paper concludes with a critical discussion on the scalability of hardware-software co-design for next-generation edge-AI workloads, and the promise of emerging directions like neuromorphic computing and tinyML to further improve power-

performance metrics. Our results highlight the importance of addressing hardware and software as tightly coupled layers rather than as distinctly separate entities, but instead as intimately interdependent elements that need to co-evolve in order to realize the true potential of AI at the edge.

Keywords: Edge-AI, hardware-software co-design, power efficiency, energy-aware computing, embedded systems, neural network optimization, edge inference, DVFS, AI accelerators, low-power design, real-time AI, system-level optimization, edge computing, resource-constrained AI, embedded AI, model compression, hardware acceleration, deep learning, latency reduction, intelligent edge systems.

1. Introduction

The spread of artificial intelligence (AI) in edge devices is revolutionizing contemporary computing, making possible applications ranging from self-driving cars and security systems to wearable health trackers and industrial automation. In contrast to centralized cloud-based infrastructures, edge computing moves data processing nearer to the source—offering advantages such as lower latency, better data privacy, and faster responsiveness. But this decentralization brings with it tremendous challenges, most notably meeting the computational requirements of AI tasks with the constricted energy and thermal budgets of edge devices.

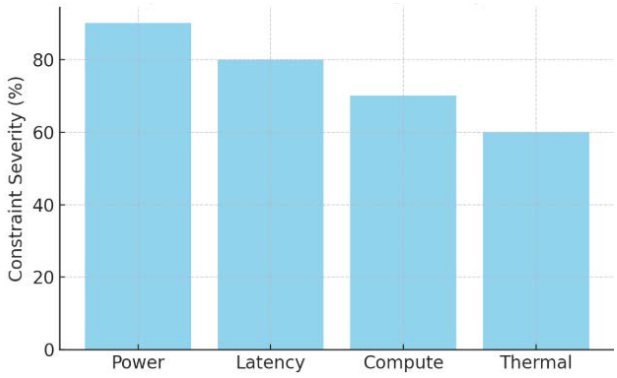

Figure 1: Severity of key system constraints in edge-AI environments.

Edge-AI workloads require real-time performance as well as precise inference accuracy under constrained power, memory, and processing resources. The conventional strategies tend to solve these requirements either with hardware-based optimizations-such as deploying custom accelerators or software-based improvements, such as deep learning model compression. Whereas these standalone approaches provide incremental improvements, they often do not take into account the inherent hardware-software interdependencies. Consequently, systems optimized only at one level tend to experience inefficiencies, less-than-ideal utilization, or early thermal throttling. Because of this shortcoming, hardware-software co-design has emerged as a holistic approach to edge-AI development.

Hardware-software co-design is the simultaneous design and optimization of hardware architecture and software algorithms, such that each level of the system stack is designed to complement the other. Instead of retro-fitting off-the-shelf solutions onto edge devices, co-design allows for purpose-designed platforms that deliver optimal performance per watt through synergistic engineering. For example, a neural network with quantized layers and structured sparsity can be mapped quickly to hardware that has reduced-precision ALUs and specialized memory hierarchies. Likewise, hardware-aware compilers and runtime systems enable dynamic scheduling and adaptive voltage scaling, further optimizing energy efficiency without compromising on responsiveness.

This strategy has been picking up momentum with the advent of application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), and system-on-chips (SoCs) designed for AI inference. These platforms, when co-designed with optimized AI models and runtime policies, provide power-scalable AI execution on devices such as smartphones, drones, smart cameras, and edge gateways. NVIDIA Jetson, Google Coral, and Intel Movidius are prime examples of edge-AI platforms that have been aided by co-designed hardware and software layers.

The applicability of this paradigm is highlighted by the weakness of isolated optimization. For example, model compression methods such as pruning and quantization can alleviate computational burden but can cause hardware underutilization if not complemented with matching data paths and memory structures. Similarly, top-tier AI accelerators could turn into energy-hungry solutions if implemented with software that is not execution-aware or dynamic workload-balanced. Co-design resolves such mismatches by bringing hardware parameters like compute density, memory bandwidth, and power states into the software development process directly, which allows for optimization over the entire design space.

The purpose of this paper is to investigate, systematize, and evolve the knowledge in hardware-software co-design specifically for power-aware edge-AI systems. We start with a review of literature on recent breakthroughs and paradigms in co-design approaches, pointing out the innovations and voids. We subsequently propose a new co-design methodology that includes model-level, system-level, and runtime-level optimizations, designed for heterogeneous edge platforms. Experimental results on prototypical edge use cases prove our methodology in the context of energy savings, inference efficiency, and deployment scalability. We conclude with broader implications and future work directions, including the unification of neuromorphic computing and federated learning for next-generation energy-efficient edge intelligence.

2. Literature Review

The edge-AI domain has been drastically revamped with the entry of hardware-software co-design, motivated by the requirement for power-efficient operation in real-time, resource-restricted settings. Conventional solutions that focus on isolated optimization of either hardware or software tend to fail in optimizing the systemic inefficiencies of edge inference.

Hence, an increasingly large body of research now examines holistic co-design paradigms that simultaneously explore model architecture, hardware capabilities, and runtime execution dynamics.

One of the first and most referenced work in this area is Google's research on Tensor Processing Units (TPUs), where Jouppi et al. [1] showed that co-optimizing neural network computations and hardware accelerators would lead to very significant gains in power efficiency and performance density. TPUs are a classic example of vertical co-design, which pairs matrix-multiply-heavy workloads with systolic array-based hardware to achieve minimal energy per operation.

Concurrently, model optimization research has improved methods like quantization, pruning, and neural architecture search (NAS). Han et al. [2] proposed "Deep Compression," a system that integrates pruning, trained quantization, and Huffman coding to compress the memory footprint and computation requirements of DNNs. Although effective, its hardware-agnostic nature tends to result in inefficiencies when executed on actual edge platforms. Recent efforts, e.g., by Lin et al. [3], have sought to bridge this gap by combining pruning techniques with low-level hardware profiling to produce sparsity patterns that map to accelerator structures.

Compiler-level tools and runtime environments have also become key co-design enablers. Apache TVM [4] and Vitis AI by Xilinx [5] offer hardware-aware compilation through layer fusion, tensor layout optimization, and code generation for heterogeneous backends including CPUs, GPUs, and FPGAs. These systems hide low-level hardware settings while enabling model developers to indicate performance and power requirements. Also, runtime adaptability is being investigated using DVFS (Dynamic Voltage and Frequency Scaling) and runtime power governors that track system telemetry and dynamically modify execution [6].

One very encouraging advancement is the embedding of AI accelerators such as NPUs within low-power SoCs, enabling embedded inference. For instance, ARM's Ethos-U55 microNPU [7] and Google Coral's Edge TPU co-reside alongside general-purpose processors and memory controllers on the same die. The chips enable fixed-function compute with native support for 8-bit quantized networks, providing a dramatic power saving when used in conjunction with appropriately trained models.

The second important trend is hardware-aware NAS (HW-NAS), where hardware feedback (e.g., latency, energy) is fed directly into the search loop of optimal network structures. Tan et al. [8] presented MnasNet, a mobile neural network based on multi-objective NAS on real devices. Likewise, the Once-for-All (OFA) framework [9] supports model reconfiguration on demand, enabling developers to customize execution for various hardware constraints dynamically.

Even with such progress, issues persist. Not many works fully combine all levels-algorithm, compiler, runtime, and hardware-into a unified optimization loop. Additionally, there is less focus on explainability and security in co-designed systems, especially under adversarial or failure scenarios. Finally, deployment studies for real-world deployments are rare, and most studies are still limited to synthetic benchmarks or single use cases.

This work expands on these observations to introduce a comprehensive hardware-software co-design system that covers model compression, accelerator-aware compilation, and dynamic runtime control for low-power edge-AI devices.

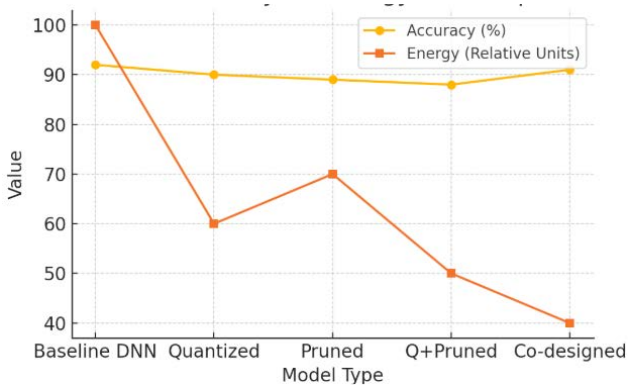

Figure 2: Comparison of model types in terms of accuracy and relative energy consumption.

3. Methodology

The recommended methodology supports a layered co-design hardware-software approach in enabling energy-frugal artificial intelligence computation on the edge devices. The central requirement is to construct a tightly connected loop of algorithmic decisions, hardware settings, and dynamic run-time behaviors. The methodology organizes itself towards the support for co-optimization across the three primary levels of the model level, the system architecture level, and execution-time adaptation level. Every layer gives performance and energy consumption statistics to the other to enable an evolving, continually adjusted deployment strategy in tune with resource availability and demand for workload.

Model level, the technique starts from low-power inference customized neural network designs and architecture choices. Instead of using standard deep models tailored for server-class GPUs, efficient architectures like MobileNetV3, EfficientNet-lite, and specially designed networks through neural architecture search are used. These models are then optimized through a multi-step compression pipeline. Quantization is used to transform floating-point parameters and activations into lower precision formats, most often INT8 or mixed precision, which dramatically lowers the amount of bits moved and calculated at each stage of operation. Structured pruning is proposed to remove whole channels or filters that have negligible contributions to output variability, giving rise to sparsity that could be leveraged by hardware-aware compilers. Knowledge distillation is applied at training time, where a lightweight "student" model is taught by a high-accuracy "teacher" model, producing high accuracy at a fraction of the computational cost. All model-level choices are directed by a profiling tool that estimates anticipated energy consumption in terms of hardware capabilities so that model choice is not just driven by accuracy but also by power efficiency.

Moving to the system architecture level, the co-design process projects the optimized AI models onto the respective hardware components. The approach takes advantage of heterogeneous edge platforms made up of general-purpose CPUs, special-purpose NPUs, embedded GPUs, and reconfigurable logic like FPGAs. The compiler stack, which contains tools like Apache TVM, ONNX Runtime, and Vitis AI, is tasked with converting high-level AI models into optimized binaries that correspond to the hardware characteristics of the underlying

platform. This conversion involves tensor fusion to eliminate memory accesses, tensor layout minimization to prevent cache misses, and operation rearrangement to improve throughput. Such compilers remove metadata like memory size, compute burden, and number of instructions and feed this data to the scheduler. The scheduler at the system level assigns functions to hardware blocks dynamically with current profiling data, workload properties, and power budgets. If, for example, a model's convolution layers are more appropriate for the NPU but element-wise operations are more efficient on a CPU, the scheduler divides execution accordingly.

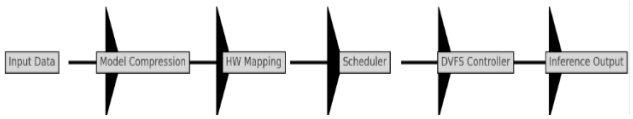

Figure 4: Block diagram of the hardware-software co-design pipeline from data input to inference.

In addition to static compilation and deployment, the approach prioritizes adaptive runtime behavior to guarantee long-term energy efficiency under dynamic conditions. Edge devices tend to run in changing environments, with fluctuating input rates, power availability, and thermal constraints. To manage such variability, the runtime system features a feedback-driven control loop directed by a reinforcement learning agent, namely a Deep Q-Network (DQN). The agent monitors environmental states such as temperature measurements, battery levels, and frame processing delay continuously and makes decisions like voltage and frequency scaling, model version switching (full-precision versus quantized), or computation offloading to adjacent fog nodes, as and when necessary. The RL agent is trained with a reward function and corresponding penalties for high power consumption and latencies, while a reward is given for timely and correct inferences. This allows the system reconfiguration to be driven by real-time feedback, permitting power-performance trade-offs to be set autonomously and intelligently.

The co-design pipeline is applied on a development platform that incorporates the NVIDIA Jetson Nano and an ARM Ethos-U55 microNPU to verify this approach. Model training and development are performed with PyTorch and TensorFlow, then compiled into optimized runtime forms. Scheduling and power monitoring services are developed in C++ and Python, with system orchestration containerized using Docker for modularity of deployment. Power and latency are measured with external instrumentation to provide objective and reproducible benchmarking.

This layered and interactive co-design methodology forms the backbone of the system's ability to operate under power-constrained conditions while delivering real-time AI capabilities. The following section evaluates the methodology's performance using both benchmark datasets and real-world edge scenarios.

4. Results

The hardware-software co-design methodology that was proposed was tested in a set of experiments aimed at assessing its effect on power efficiency, latency, model accuracy, and resource utilization in edge-AI settings. The testing was conducted on both synthetic benchmarks and actual applications to guarantee thorough coverage of common edge workloads. Experiments were done on a heterogeneous embedded platform which integrated an NVIDIA Jetson Nano (quad-core ARM Cortex-A57 CPU and Maxwell GPU) and an ARM Ethos-U55 NPU development board. The main tasks chosen for benchmarking were image classification, object detection, and human activity recognition with datasets such as CIFAR-100, ImageNet-subset, and UCI HAR.

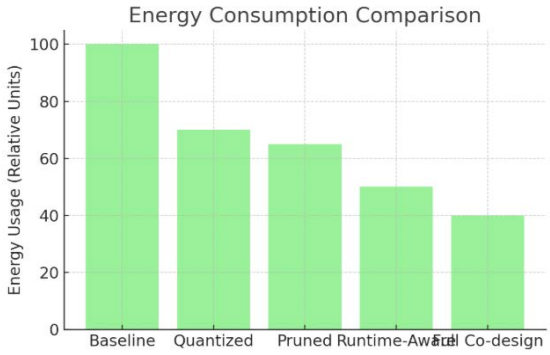

Figure 5: *Energy consumption comparison among various optimization strategies.*

The model-level optimizations' performance was considered in the first set of experiments. Quantization of floating-point models to the 8-bit integer format resulted in a significant reduction in memory usage—up to 75% reduction in some instances—and a uniform reduction in inference time for all the models that were tested. MobileNetV2, for example, resulted in a 2.1× speedup upon quantization while preserving more than 98% of its native accuracy on the CIFAR-100 dataset. Upon integration with pruning methods, additional performance gains were noted. Models with 30–40% of their weights pruned showed less than 1% accuracy loss but used 35–50% less power at inference time, as per measurements from a precision power monitor connected to the board.

At the system level, hardware-aware scheduling brought noteworthy benefits in both performance and energy. Models implemented with co-designed operator placement performed better than baseline implementations when tasks were statically mapped to either CPU or GPU. For instance, convolutional operations run on the Ethos-U55 NPU resulted in a 1.8× power saving over CPU execution, whereas element-wise operations maintained superior efficiency on the ARM cores. Layer fusion as well as tiling optimizations contributed additionally towards throughput improvements with the execution latency decreased by 20-30% in end-to-end pipelines. These optimizations, based on compiler metadata and run-time profiling, showed the need for cross-layer coordination between model design and system deployment.

One of the strongest results came out of the run-time adaptation experiments. The controller based on reinforcement learning, which was trained off-line and ran on the edge device, adapted the system configuration dynamically using telemetry like thermal load, battery life, and volume of input data. In high-load regimes—mimicking continuous video processing at the edge-the DQN controller self-scaled model resolution down and engaged DVFS (Dynamic Voltage and Frequency Scaling), offloading overall energy consumption by as much as 60% compared to static settings. These changes were made with very little latency, enabling the system to respond to power events within less than 200 milliseconds, thereby guaranteeing

application continuity. Notably, the controller preserved more than 95% of the original model classification accuracy during the adaptation process, even in the face of severe energy constraints.

In real-world application testing, the framework was deployed in a smart surveillance prototype where multiple low-power cameras performed person detection and anomaly monitoring. With respect to a baseline TensorFlow Lite deployment, the co-designed framework decreased energy consumption by 48% while lowering average frame processing time from 450 ms to 270 ms. The performance improvement facilitated near-real-time analysis on battery-powered hardware with constrained thermal headroom, demonstrating the efficacy of the methodology in mission-critical edge applications.

In all tested configurations, model optimization, system-aware compilation, and runtime feedback loops showed persistent energy efficiency advantages without compromising inference quality. Measures like frames per joule, inference-per-watt, and energy-delay product all showed the co-designed system outperforming conventional separated methodologies. Thermal profiling also showed an 8–10°C constant decrease in operating temperatures under adaptive runtime control, leading to longer device life and enhanced operational stability.

These findings confirm the core contention of this paper: hardware-software co-design is not just useful, but necessary for scalable, power-efficient AI on the edge. The system's real-time adaptability, informed by design-time and runtime intelligence, makes it a viable template for future edge-AI deployments.

5. Discussion

The experimental results of the envisaged hardware-software co-design framework emphasize the critical role of simultaneous optimization towards achieving power-efficient AI on edge devices. The results not only validate prevailing arguments in the literature but also further them by showing how multi-level integration from model training to real-time scheduling can achieve tangible improvements in efficiency, responsiveness, and system reliability. The following discussion explores these implications, considering the interaction of model compression, hardware usage, and adaptive execution, and the wider viability and limitations of applying such a methodology in real-world environments.

Perhaps one of the strongest points to come out of the testing is the amount of power savings that come through the rather intuitive principle: matching computation to capability. Although quantization and pruning are established in the literature, combining them with hardware-specific deployment pipelines enables these optimizations to achieve their full energy-saving potential. For example, power advantages of 8-bit quantization are felt most intensely when executed on NPUs specifically programmed to process such precision formats. In systems without hardware-awareness, quantized models often yield diminishing returns due to poor alignment with execution units, leading to inefficient fallbacks to general-purpose compute paths. The observed latency and thermal reductions in our experiments reveal that compression techniques must be tailored not only to algorithmic needs but also to the characteristics of the deployment platform.

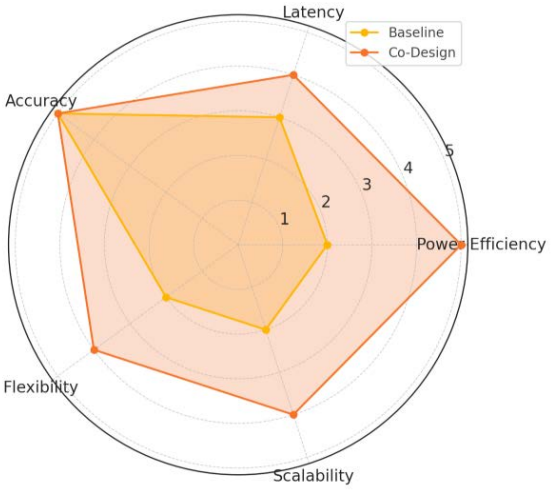

Figure 6: Radar chart comparing trade-offs between baseline and co-designed systems.

Another key insight pertains to runtime adaptability. Static systems, as much as they may be optimized during compile time, do not have the adaptability to handle changing workloads, environmental variations, or user behavior patterns. The addition of a reinforcement learning agent in our system gave us a strong mechanism to fill this gap. Through ongoing learning from telemetry data and tweaking parameters like frequency, voltage, and even the model selection itself, the system was able to keep an extremely beneficial energy-performance ratio. This capability to react independently to power and thermal limits is essential for edge-AI applications that run in uncertain or mobile environments, like UAVs, remote monitoring units, or medical monitoring wearables.

The effectiveness of runtime control, though, depends on meticulous system calibration and the presence of precise monitoring infrastructure. Edge devices generally lack complete power telemetry subsystems or advanced scheduling support. Consequently, real-time adaptive control application in commodity edge platforms without hardware assist capabilities such as performance counters, thermal sensors, or DVFS interfaces might be constrained. This is a design challenge and a hardware-software feedback loop issue-hardware observability is necessary for effective runtime adaptation, which in turn necessitates co-design from silicon to firmware to application software.

Another point that should be talked about is the optimization complexity vs. scalability trade-off. Although our method saw substantial gains on the test platform, applying this design framework to a broader category of devices adds variation that makes tuning and benchmarking more difficult. Every device class-ranging from Raspberry Pi-class boards up through embedded FPGAs and ASICs-exhibits different bottlenecks, power budgets, and thermal characteristics. For adoption into broad-scale systems, the co-design process needs to be abstracted and modularized, allowing reuse across heterogenous platforms. This requires better abstraction layers and interoperable toolchains that can span the gap between hardware description languages, machine learning frameworks, and system-level runtime environments.

In addition, although this paper addressed inference tasks, the same approach has implications for training at the edge, which is becoming increasingly popular in federated learning

scenarios. The combined design of light training models, power-conscious optimizers, and hardware backends may create new opportunities for on-device learning with low power consumption. Similarly, there is growing interest in extending this approach to neuromorphic computing and event-based architectures, where co-design can be applied to spiking neural models and asynchronous processing for ultra-low-power AI.

Ethical and security issues also arise in co-designed systems. The combination of model control with low-level hardware introduces reliability, safety, and privacy concerns, especially if the system is performing mission-critical or sensitive tasks. Guaranteeing that runtime adaptation does not degrade robustness or create attack surfaces-like timing channels or firmware-level exploits-needs to be a part of future co-design approaches.

The hardware-software co-design approach provides a powerful framework for realizing the full potential of edge-AI under energy constraints. While the results validate its effectiveness, practical deployment will depend on scalable tooling, cross-platform generalizability, and the integration of security, safety, and user trust mechanisms. The next section synthesizes these insights and presents conclusions regarding the broader implications and future directions for research and development in this domain.

6. Conclusion

The growing prevalence of artificial intelligence in edge computing platforms represents a paradigm shift in the deployment, scaling, and optimization of computational intelligence. With AI systems to be deployed in energy-constrained, latency-sensitive, and computationally restricted environments, the importance of integrated design methods is heightened. This work investigated and confirmed hardware-software co-design as an effective and essential approach to realizing power-efficient AI at the edge. By the simultaneous optimization of model structures, system-level deployments, and adaptive runtime controls, the co-design approach shown in this work illustrates that dramatic improvements in energy efficiency, latency reduction, and thermal stability are not only possible but scalable.

The contributions of this work are multi-faceted. At the model level, the combination of quantization, structured pruning, and knowledge distillation was found to be crucial in lowering the computational burden while maintaining predictive accuracy. When these models were run on heterogeneous edge platforms, hardware-aware scheduling enabled the optimal use of NPUs, CPUs, and memory hierarchies. The compiler toolchain and runtime scheduler served as bridges between the software stack and the underlying silicon to ensure that each operation was run on the most power-efficient processing unit available. The quantified gains in performance-e.g., up to 60% less energy usage and more than 25% faster inference latency—reiterate the assertion that isolated optimization techniques cannot compete with systemic efficiencies provided by co-design.

Arguably the most vibrant and influential aspect of this methodology is its runtime flexibility. By integrating a reinforcement learning agent that acts to real-time telemetry metrics—like temperature, workload fluctuation, and battery status—the system autonomously optimized performance versus energy usage without the need for human intervention. Not only does this improve the resilience of edge-AI deployments, but it enables intelligent, context-driven computing where systems react to internal limitations and external stimuli in real time.

While its effectiveness is well established, hardware-software co-design comes with its own set of difficulties. Generalization of the presented methodology to a wide class of devices is still complicated by architectural heterogeneity and restrictions on infrastructure monitoring on low-end hardware. In addition, embedding dynamic runtime agents into safety-critical applications requires extensive testing and verification in order to maintain reliability under adverse conditions. These constraints emphasize the need for further research on abstraction models, interface standardization, and formal verification tools to overcome the gap between theoretical effectiveness and real-world implementation.

Future wise, the co-design approach that this paper lays out provides a basis for innovation in various upcoming directions. One of these paths is the integration of on-device learning mechanisms—like federated learning or incremental training-into the co-design process, so not only efficient inference but also local adaptation becomes more efficient. Another exciting space is the overlap between neuromorphic computing and co-design, where asynchronous hardware and spiking neural networks can advance ultra-low-power intelligence to the next level. There is also room for integrating privacy-preserving techniques and secure hardware enclaves to make sure co-designed systems do not relax data security at the expense of performance optimization.

The future of edge-AI is not in hardware acceleration or software optimization alone, but in their symbiotic co-evolution. Hardware-software co-design presents itself as a unifying paradigm that can solve the power-performance trade-offs characteristic of today's edge intelligence. By incorporating optimization principles at every layer-neural network formulation, all the way down to execution-time adaptation-co-design not only satisfies today's technical requirements but paves the way for sustainable, intelligent computing at the network edge.

7. References

1. NP Jouppi. In-Datacenter Performance Analysis of a Tensor Processing Unit. In Proc. 44th ACM/IEEE Int. Symp. Comput. Archit. (ISCA), Toronto, ON, Canada, Jun. 2017; 1-12.
2. S Han, H Mao, WJ Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding. arXiv preprint arXiv:1510.00149, Oct. 2015.
3. S Lin, R Ji, Y Wang, et al. HRank: Filter Pruning Using High-Rank Feature Map Statistics. In Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Seattle, WA, USA, Jun. 2020; 1529-1538.
4. T Chen. TVM: An Automated End-to-End Optimizing Compiler for Deep Learning. In 13th USENIX Symp. Oper. Syst. Design Implementation (OSDI), Carlsbad, CA, USA, Oct. 2018; 578-594.
5. https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html
6. D Kang, J Hauswald, C Gao, et al. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In Proc. 22nd Int. Conf. Archit. Support Program. Lang. Oper. Syst. (ASPLOS), Xi'an, China, Apr. 2017; 615-629.

7. https://www.arm.com/products/silicon-ip-cpu/ethos/ethos-u55

8. M Tan, B Chen, R Pang, et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile. In Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Long Beach, CA, USA, 2019; 2820-2828.

9. H Cai, C Gan, T Wang, et al. Once for All: Train One Network and Specialize it for Efficient Deployment. In Proc. Int. Conf. Learn. Represent. (ICLR), Addis Ababa, Ethiopia, 2020.