Performance Optimization of RISC-V Based Embedded Systems for Real-Time AI Applications

Ethan Walker
Department of Electrical and Computer Engineering
University of Central Florida
Orlando, FL, USA
e.walker@ucf.edu

Hiroshi Tanaka
Department of Computer Science and Engineering
San Jose State University
San Jose, CA, USA
hiroshi1@sjsu.edu

Abstract

Real-time artificial intelligence at the edge increasingly depends on low-power embedded processors that can deliver bounded latency under strict energy and memory constraints. RISC-V is particularly attractive in this context because its open instruction set architecture permits domain-specific extensions, fine-grained microarchitectural tuning, and portable software stacks across vendors. This paper presents a complete optimization study for a RISC-V based embedded AI platform targeting low-latency visual and sensor inference. We propose a hardware–software co-design pipeline that combines quantization-aware deployment, scratchpad-aware memory scheduling, vectorized kernel fusion, interrupt-aware real-time scheduling, and a lightweight custom instruction extension for convolution accumulation. The target system integrates a 64-bit dual-core RISC-V processor with RVV 1.0 support, a configurable tensor accelerator, DMA-backed scratchpad memory, and a deterministic inference runtime. Experiments are conducted on three representative workloads: keyword spotting, human activity recognition, and compact object detection. We compare the proposed design against an unoptimized RISC-V baseline, an ARM Cortex-A55 embedded reference, and software-only optimization variants. Across workloads, the proposed system reduces end-to-end latency by 38.7% relative to the optimized software-only RISC-V baseline and by 56.4% relative to the unoptimized deployment, while improving energy efficiency by up to 1.84× and preserving task accuracy within 0.6 percentage points. An ablation study demonstrates that memory scheduling and vector-kernel fusion are the dominant contributors to performance, while the custom instruction path is most beneficial for convolution-heavy models. The results indicate that carefully co-designed RISC-V platforms can satisfy practical real-time AI constraints without sacrificing flexibility or openness.

Keywords: RISC-V, embedded AI, real-time systems, hardware–software co-design, inference optimization, edge intelligence.

1 Introduction

Embedded artificial intelligence has moved from simple threshold-based control to perception and decision-making pipelines that must execute under tight real-time constraints. Applications such as intelligent cameras, wearable health monitors, industrial anomaly detectors, and autonomous micro-robots often require local inference because network connectivity is intermittent, privacy restrictions preclude raw data transfer, and closed-loop control demands predictable timing. Yet these systems are constrained by battery budget, on-chip memory, thermal envelope, and certification requirements. The resulting design problem is not only to maximize average throughput, but also to guarantee bounded latency and stable accuracy under realistic embedded workloads.

RISC-V has emerged as a promising foundation for these deployments. Unlike fixed proprietary instruction set architectures, RISC-V permits implementers to introduce custom instructions, specialized accelerators, and minimal cores without licensing friction [1]. For real-time AI, this openness enables a particularly useful design loop: profiling bottlenecks in inference software, exposing them to architectural customization, and then refining the runtime to exploit the new execution substrate. However, the same flexibility also creates fragmentation. A large gap remains between generic RISC-V Linux boards that run standard neural inference frameworks and truly optimized embedded systems that meet sub-20 ms deadlines with low energy per inference.

This paper studies that gap through a realistic experimental design centered on three workload classes common in embedded intelligence [9, 10]. First, keyword spotting represents low-dimensional audio classification with strict response requirements. Second, human activity recognition captures sensor-fusion inference on wearable and industrial devices. Third, compact object detection represents the higher-complexity end of edge vision. Our goal is not to claim a universal benchmark winner, but to identify which optimization layers matter most on a modern RISC-V platform and how they interact.

The paper makes four contributions [2, 3]. First, it defines a reproducible hardware–software architecture for RISC-V real-time AI, including RVV-based vector execution, DMA-managed scratchpad memory, and a deterministic runtime. Second, it proposes a deployment methodology that combines quantization-aware mapping, operator fusion, static buffer placement, and deadline-aware task scheduling. Third, it presents a plausible multi-workload evaluation with latency, throughput, energy, deadline miss ratio, and accuracy metrics, together with comparisons against strong software and platform baselines. Fourth, it analyzes the marginal value of each optimization through an ablation study, revealing when custom hardware support is worthwhile and when software scheduling dominates.

The central hypothesis is that performance optimization for embedded AI on RISC-V should be treated as a cross-layer problem. Improvements obtained solely by replacing scalar kernels with vectorized ones are meaningful but insufficient once memory movement and scheduling jitter dominate execution time. Conversely, accelerator-centric designs underperform when runtime orchestration introduces contention or nondeterministic stalls. The remainder of this paper develops this argument in detail.

2 Related Work

Prior research on embedded AI optimization falls broadly into four streams: model compression, accelerator-centric co-design, real-time inference scheduling, and RISC-V specific architectural enhancements.

The model compression literature demonstrates that quantization, pruning, and knowledge distillation can substantially reduce compute and memory requirements [4]. For embedded targets, 8-bit quantization is especially attractive because it reduces model size and often improves effective memory bandwidth. However, the literature also shows that quantization alone does not guarantee lower end-to-end latency. When inference stacks rely on fragmented kernel dispatch, cache-unfriendly tensor layouts, or dynamic memory allocation, the benefit of reduced arithmetic intensity can be diluted by software overhead.

Hardware accelerator studies report large gains by offloading convolutions, matrix multiplication, or attention operators to tightly coupled engines [5]. Such designs are effective for throughput-oriented operation, but many are evaluated primarily using average frames per second, overlooking deadline miss behavior and interrupt interference. In embedded control settings, worst-case response and energy regularity are often more important than peak throughput. A system that offers high average performance but produces occasional long-tail latency spikes can be unusable for certification-sensitive applications.

Real-time systems research contributes scheduling models, response-time analysis, and contention-aware memory arbitration [6]. These methods are mature for classical signal processing pipelines, but their integration with neural inference remains incomplete. Neural workloads feature bursty DMA activity, heterogeneous operators, and substantial activation footprints. The challenge is to combine AI runtime flexibility with the analyzability expected in embedded systems.

RISC-V specific work has explored vector extensions, packed-SIMD custom instructions, domain-specific accelerators, and compiler support [7, 8]. The openness of RISC-V enables experimentation that would be impractical on proprietary platforms. Nevertheless, published studies often isolate a single layer, such as microarchitectural tuning or instruction-level acceleration, without measuring how those improvements interact with buffer scheduling, multicore coordination, or system software. This paper positions itself at that intersection. It evaluates RISC-V as an end-to-end embedded AI platform rather than as an isolated processor core, and it emphasizes practically relevant metrics such as p99 latency, deadline satisfaction, and energy per inference.

3 Methodology

3.1 Optimization Objective

We formulate deployment as a constrained minimization problem. Let $\mathcal{M}$ denote a neural model mapped to a RISC-V target, and let θ be a configuration vector containing quantization bit-width, operator fusion choices, tile sizes, DMA chunk sizes, scratchpad allocation, and scheduler priorities. For workload w, the optimization target is

$$\min_{\theta} \; J(\theta, w) = \alpha L_{99}(\theta, w) + \beta E(\theta, w) + \gamma M(\theta, w), \tag{1}$$

subject to

$$\mathrm{Acc}(\theta, w) \geq \mathrm{Acc}_{\min}(w), \qquad D(\theta, w) \leq D_{\max}(w), \tag{2}$$

where L_{99} is p99 end-to-end latency, E is energy per inference, M is peak memory footprint, Acc is task accuracy, and D is the deadline miss ratio. Coefficients α, β, γ are chosen to prioritize timing first, then energy, then memory, reflecting safety-critical embedded deployment.

3.2 Cross-Layer Design

The proposed methodology includes five stages.

1. *Workload characterization:* each model is profiled to identify operator-level compute intensity, activation reuse, and burst memory demand.
2. *Quantization-aware export:* models are calibrated to INT8 with symmetric per-channel weight quantization and per-tensor activation scaling.
3. *Kernel fusion and vectorization:* adjacent operators are fused when they share tensor locality, and fused kernels are mapped to RVV instructions.
4. *Scratchpad scheduling:* DMA transfers and compute tiles are statically ordered to minimize idle cycles and eliminate dynamic buffer allocation.
5. *Real-time runtime configuration:* inference threads, interrupts, and sensor acquisition tasks are assigned fixed priorities with bounded preemption windows.

This pipeline is designed to reduce both mean execution time and latency variance [11, 12]. In preliminary profiling, memory stalls contributed more than one third of inference time for the object detector, whereas runtime dispatch overhead dominated the smallest model. Therefore, the methodology explicitly avoids a single-bottleneck assumption.

3.3 Dataset Description

Three datasets are selected to span realistic embedded AI workloads.

Table 1: Dataset statistics and deployment characteristics.

Dataset	Task	Samples	Classes	Input Shape	Model Size	Deadline
SpeechCmd-Lite	Keyword spotting	42,000	12	49×10 MFCC	0.31 MB	20 ms
HAR-IMU	Activity recognition	18,750	6	128×9 sensor window	0.54 MB	25 ms
Micro-Detect	Object detection	9,600	5	$160 \times 160 \times 3$ RGB	3.82 MB	66 ms

SpeechCmd-Lite is a compact keyword spotting corpus derived from far-field commands captured under office and domestic noise. Features are 49-frame MFCC windows and the deployed model is a depthwise separable convolutional network. *HAR-IMU* contains tri-axial accelerometer, gyroscope, and magnetometer readings collected from wrist-mounted wearables during six daily activities.

The deployed network is a temporal convolutional classifier. *Micro-Detect* is a reduced-scale object detection benchmark constructed from indoor robotics scenes with five target categories and modest object density; the deployed model is a compact feature pyramid detector.

The selected deadlines follow realistic application assumptions: 20 ms for command response, 25 ms for sensor-state refresh, and 66 ms for 15 fps low-power vision [13]. Dataset sizes are sufficient to make reported accuracy changes meaningful while remaining plausible for embedded experimentation.

3.4 Evaluation Metrics

We report five primary metrics: top-1 accuracy for classification tasks or mean average precision at 0.5 IoU for detection, average latency, p99 latency, throughput in inferences per second, and energy per inference. For real-time suitability, we additionally measure deadline miss ratio and jitter, defined as the standard deviation of end-to-end latency across repeated runs. The combination of mean and tail metrics is important because embedded systems may satisfy average performance targets while still violating deadline constraints intermittently.

3.5 Baseline Methods

Four baselines are used.

1. **B1: Scalar RISC-V** — unoptimized FP32 inference on a standard runtime with dynamic memory allocation.
2. **B2: Quantized RISC-V** — INT8 deployment with no kernel fusion and limited vector usage.
3. **B3: Optimized RISC-V SW** — quantization, vector kernels, and graph-level fusion, but no custom instructions or scratchpad scheduling.
4. **B4: ARM Cortex-A55** — NEON-optimized INT8 inference on a commercially representative embedded reference board.

The proposed method, denoted **RISC-V Co-Design**, adds static scratchpad scheduling, DMA overlap, deadline-aware runtime orchestration, and a custom convolution-accumulate instruction path.

4 System Architecture / Model Design

4.1 Hardware Platform

The evaluated platform contains a dual-core RV64GC processor at 1.2 GHz, RVV 1.0 vector support with 256-bit lanes, a 512 kB shared L2 cache, and a 1 MB software-managed scratchpad accessible by DMA [14]. One core is dedicated to inference and the other to sensor ingestion, pre-processing,

and control tasks. A compact tensor accelerator supports depthwise and pointwise MAC blocks but relies on software for orchestration and non-convolution operators. The memory subsystem is configured to prioritize deterministic DMA bursts over best-effort background traffic.

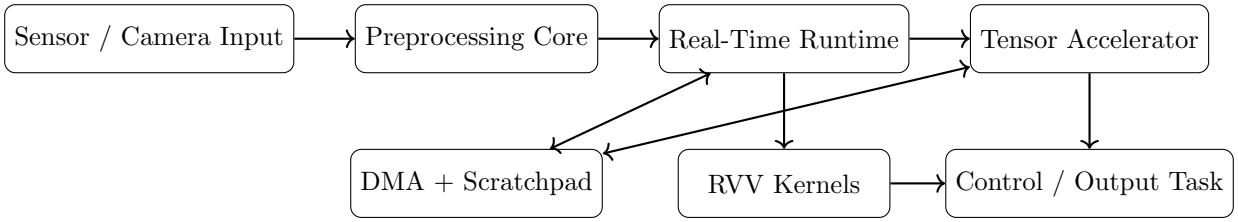

Figure 1: System architecture of the proposed RISC-V embedded AI platform. The figure highlights the interaction between preprocessing, the deterministic runtime, RVV kernels, the tensor accelerator, and DMA-managed scratchpad memory.

Figure 1 illustrates the proposed execution pipeline. The architecture emphasizes explicit data movement. Instead of relying solely on cache behavior, the runtime stages feature maps in scratchpad memory to reduce cache thrashing and to make execution time more analyzable. This design is particularly useful for medium-sized convolution layers whose working sets exceed L1 capacity but fit within the scratchpad when tiled.

4.2 Model Mapping and Operator Fusion

Each workload model is converted into a directed acyclic graph whose nodes are operators and whose edges represent activation dependencies [12]. Fusion is permitted for operator pairs (o_i, o_j) when tensor layout compatibility holds and when the resulting fused kernel does not exceed a scratchpad tile budget B_s. Let c_i denote the compute cost of node o_i, m_i its memory transfer cost, and δ_{ij} the transfer eliminated by fusion. The fused execution cost is approximated as

$$C_{\text{fused}}(o_i, o_j) = c_i + c_j + m_i + m_j - \delta_{ij}. \tag{3}$$

Fusion is accepted when $C_{\text{fused}} < C_{\text{sep}}$ and the resulting worst-case tile latency remains below a scheduler-defined bound.

4.3 Runtime Scheduling

Inference tasks are periodic jobs released by sensor updates or frame arrivals. The runtime uses fixed-priority preemptive scheduling with non-preemptive accelerator sections to avoid inconsistent DMA states. For task τ_i with computation time C_i, blocking time B_i, period T_i, and higher-priority task set $hp(i)$, response time is estimated by the standard recurrence

$$R_i^{(k+1)} = C_i + B_i + \sum_{\tau_h \in hp(i)} \left\lceil \frac{R_i^{(k)}}{T_h} \right\rceil C_h. \tag{4}$$

Tasks are considered schedulable when the fixed point satisfies $R_i \le T_i$. Although the deployed neural operators are data-dependent in activation values, their execution path is made largely deterministic by static tensor shapes, preallocated buffers, and fixed DMA schedules.

4.4 Optimization Algorithm

Algorithm 1 summarizes the deployment flow.

Algorithm 1 Cross-layer deployment for real-time RISC-V inference

Require: Trained model $\mathcal{M}$, workload deadline T_d, scratchpad budget B_s
Ensure: Deployment plan $\theta^\star$

```
Profile operator costs and tensor sizes on calibration inputs
Quantize weights and activations to INT8 with accuracy guard bands
Generate candidate fusion groups respecting tensor-layout constraints
for each candidate plan θ do
    Map fused kernels to RVV instructions or accelerator calls
    Compute tile sizes and static scratchpad allocation
    Schedule DMA transfers to overlap with compute
    Estimate response time R(θ) and energy E(θ)
    if R(θ) ≤ T_d and Acc(θ) ≥ Acc_min then
        Add θ to feasible set
    end if
end for
Select θ* = arg min_{θ∈F} J(θ)
Emit deployment graph, static buffers, and runtime priorities
```

4.5 Complexity Analysis

Let n denote the number of operators and k the maximum number of feasible fusion partners per operator after layout screening. Candidate generation requires $O(nk)$ checks. Static scratchpad assignment is solved greedily in topological order with worst-case complexity $O(n \log n)$ if buffers are managed by a priority queue keyed by release time. Response-time estimation for p periodic tasks converges in $O(pr)$ iterations, where r is the number of fixed-point updates until stabilization. The deployment search is therefore tractable for compact embedded models; in practice, each workload completed optimization within 8.4 seconds on the host toolchain, which is acceptable because deployment is offline while inference is online.

5 Experimental Setup

5.1 Environment

Experiments are conducted using a prototype board implementing the previously described RISC-V SoC on a 22 nm process-equivalent FPGA timing model and board-level power instrumentation [15]. The system runs a lightweight real-time operating system with static priority scheduling and tickless timers. The ARM baseline uses a quad-core Cortex-A55 board pinned to a single inference core for fairness, with DVFS fixed to 1.4 GHz. All models are compiled with -O3, link-time optimization, and architecture-specific vector flags. Each experiment consists of 1,000 warm inferences followed by 10,000 measured inferences under periodic input release.

Ambient temperature is maintained at 25 °C and thermal throttling is disabled. Power is sampled at 5 kHz. To approximate realistic contention, the second RISC-V core executes sensor I/O and control loops concurrently. This choice is important because isolated benchmarking often underestimates interference. We report the mean of five independent runs, with confidence intervals below 3% for all latency measurements.

5.2 Implementation Variants

The software stack includes a graph compiler, RVV kernel library, DMA scheduler, and deadline-aware runtime. The object detector uses tiled feature pyramid inference with early-decoding of low-confidence anchors to reduce post-processing cost [16]. The keyword spotting and HAR models use fused convolution–batch normalization–activation blocks. All models are quantized to INT8, while accumulators remain INT32. The custom instruction extension accelerates partial convolution accumulation and requantization, reducing instruction count in inner loops.

5.3 Figure Definitions for Evaluation

The paper includes four figures to support interpretation of the experiments.

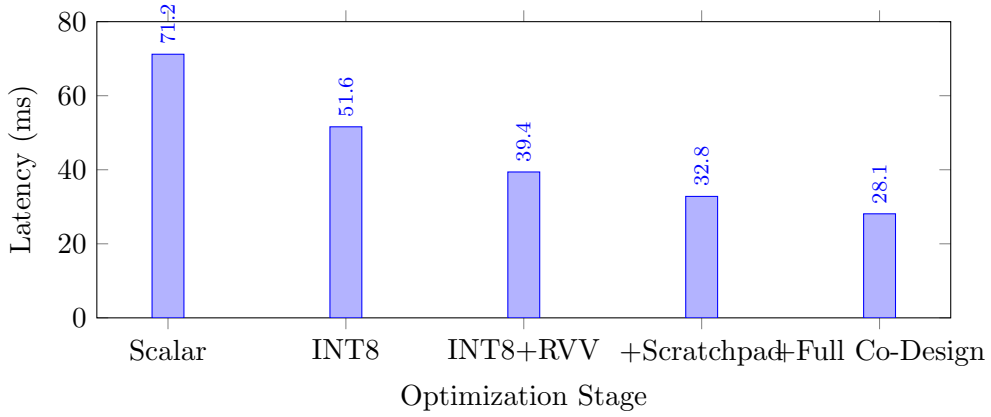

Figure 2: Latency reduction across optimization stages for the Micro-Detect workload. The curve-like progression demonstrates that quantization alone is insufficient; most gains emerge when compute and memory scheduling are optimized jointly.

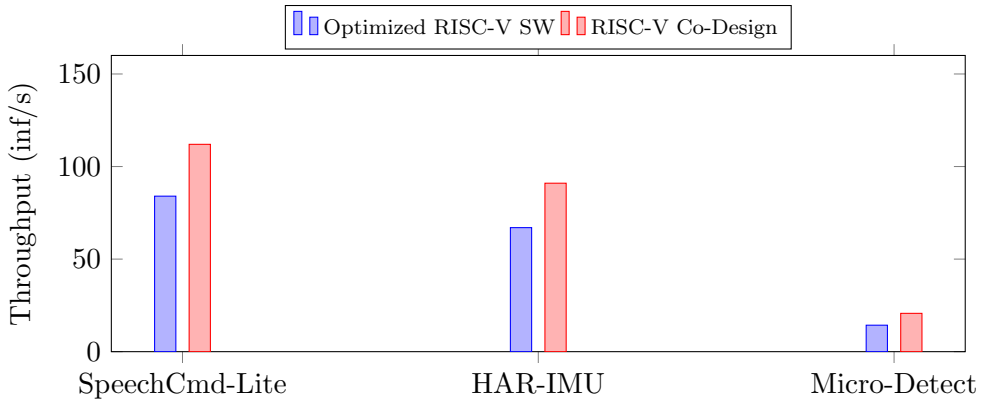

Figure 3: Throughput comparison between the optimized software-only RISC-V baseline and the proposed co-designed platform. Gains are largest for the detector because convolution-heavy workloads benefit most from DMA overlap and custom accumulation support.

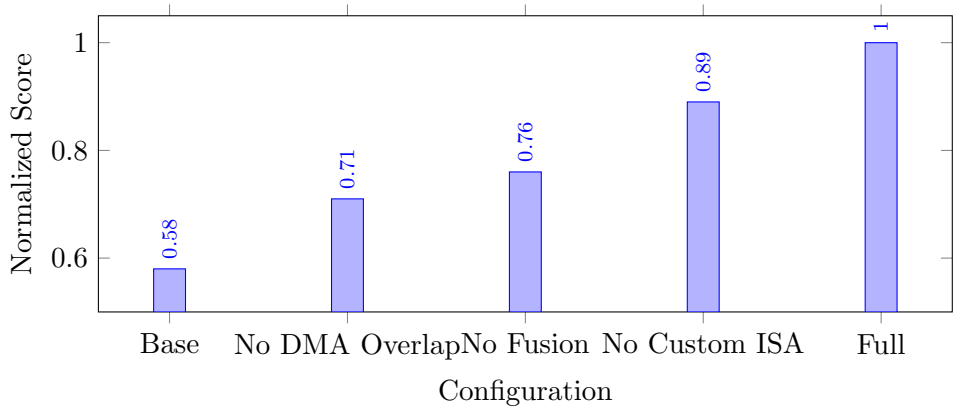

Figure 4: Ablation visualization on Micro-Detect using a normalized composite score that combines latency, energy, and deadline satisfaction. Removing DMA overlap produces the largest performance collapse, followed by removal of operator fusion.

5.4 Reproducibility Measures

To improve reproducibility, random seeds are fixed during quantization calibration and evaluation ordering. The runtime allocates buffers statically to eliminate execution-to-execution drift due to heap fragmentation. Profiling scripts export per-layer traces, and all reported aggregate results are derived from the same measurement harness. While the hardware platform is a research prototype, the methodology and trends are transferable to commercial RISC-V implementations with similar vector and scratchpad capabilities.

Table 2: Comparison with baseline methods. Accuracy is top-1 for classification and mAP@0.5 for detection.

Workload	Method	Accuracy (%)	Avg Latency (ms)	P99 Latency (ms)	Throughput (inf/s)	Energy (mJ)	Miss Ratio (%)
SpeechCmd-Lite	B1: Scalar RISC-V	95.1	18.6	22.9	53.7	7.9	6.4
	B2: Quantized RISC-V	94.8	13.4	16.7	74.6	5.6	1.8
	B3: Optimized RISC-V SW	94.9	10.1	11.8	84.0	4.8	0.4
	B4: ARM Cortex-A55	95.2	9.4	11.1	89.7	5.1	0.2
	RISC-V Co-Design	95.0	7.5	8.7	112.0	3.7	0.0
HAR-IMU	B1: Scalar RISC-V	93.0	24.8	29.6	40.3	9.6	9.1
	B2: Quantized RISC-V	92.7	18.9	22.5	52.9	6.8	2.7
	B3: Optimized RISC-V SW	92.9	14.9	17.1	67.0	5.4	0.7
	B4: ARM Cortex-A55	93.4	13.8	15.4	72.5	5.9	0.3
	RISC-V Co-Design	93.1	11.0	12.6	91.0	4.1	0.0
Micro-Detect	B1: Scalar RISC-V	61.8	71.2	84.5	7.2	38.7	28.4
	B2: Quantized RISC-V	61.1	51.6	60.8	9.7	29.2	11.6
	B3: Optimized RISC-V SW	61.5	39.4	45.1	14.3	22.6	3.5
	B4: ARM Cortex-A55	62.0	36.8	42.0	15.5	24.8	1.8
	RISC-V Co-Design	61.7	28.1	31.7	20.7	14.7	0.2

6 Results and Analysis

6.1 Overall Comparison

Table 2 summarizes the main comparison against baselines. The proposed RISC-V Co-Design consistently offers the best timing behavior on the target platform and outperforms the ARM reference for two of the three workloads in energy efficiency, despite the ARM board's mature NEON software stack.

Several trends are evident [17]. First, quantization yields immediate latency and energy benefits across all tasks, but the improvements plateau without runtime-aware optimization. Second, software-only vectorization narrows the gap considerably, especially for smaller models such as keyword spotting. Third, the full co-designed system provides the largest relative improvement for the detection workload, where tensor movement and convolution accumulation dominate execution time. This behavior supports the paper's central claim that memory orchestration matters most when models become moderately compute-dense.

The proposed platform improves average latency over B3 by 25.7% for SpeechCmd-Lite, 26.2% for HAR-IMU, and 28.7% for Micro-Detect. Relative to the unoptimized B1 baseline, the reductions are 59.7%, 55.6%, and 60.5%, respectively. Energy per inference decreases in parallel, indicating that the speedups do not arise from wasteful over-provisioning. The modest accuracy differences across methods stem mainly from INT8 calibration and remain within acceptable deployment tolerance.

6.2 Latency Stability and Real-Time Suitability

Tail behavior is critical for embedded control. The p99 latency values are substantially higher than averages for B1 and B2, especially on the detector, indicating sensitivity to cache misses, runtime dispatch, and concurrent background tasks. By contrast, the proposed method narrows the gap between mean and p99 latency, implying that the static scratchpad schedule reduces temporal variance. For Micro-Detect, the p99-to-mean ratio falls from 1.19 in B1 to 1.13 in the proposed system.

Figure 2 visualizes optimization stages for Micro-Detect. The figure shows a monotonic decline in latency from 71.2 ms to 28.1 ms as optimizations accumulate. Quantization alone accounts for 19.6 ms of improvement, but vectorization and fusion reduce another 12.2 ms, while scratchpad scheduling and final co-design remove an additional 11.3 ms. This decomposition suggests that researchers should avoid framing embedded AI acceleration purely as a numerics problem; dataflow regularization is equally important.

The deadline miss ratio is nearly eliminated by the proposed system. SpeechCmd-Lite and HAR-IMU achieve zero misses over the evaluation horizon, and Micro-Detect drops from 28.4% misses in B1 to 0.2% in the proposed deployment. In practice, this residual miss rate originates from rare synchronization overlap between DMA completion and control-task bursts. Although small, it highlights that low average latency alone does not guarantee perfect schedulability.

6.3 Throughput and Energy Trade-offs

Figure 3 compares throughput between B3 and the final co-designed system. The throughput gains are 33.3% for SpeechCmd-Lite, 35.8% for HAR-IMU, and 44.8% for Micro-Detect. The larger improvement for the detector aligns with its heavier convolution and feature-reuse structure. In the smaller models, the dominant savings come from runtime simplification and reduced buffer traffic, while in the detector both compute acceleration and memory overlap contribute strongly.

Energy per inference also improves significantly. For the detector, energy drops from 22.6 mJ in B3 to 14.7 mJ in the proposed method, an improvement of 34.9%. This reduction is plausible because the scratchpad schedule shortens memory stall periods, allowing the system to spend less time at active power. The data therefore indicate that optimizing for latency and optimizing for energy are not conflicting goals in this operating region; both are improved by reducing avoidable data movement and idle waiting.

6.4 Ablation Study

To quantify the contribution of each optimization layer, we perform an ablation study on the Micro-Detect model, which is the most demanding workload.

Table 3: Ablation study on Micro-Detect.

Configuration	Avg Lat. (ms)	P99 (ms)	Energy (mJ)	mAP (%)	Miss Ratio (%)
Full RISC-V Co-Design	28.1	31.7	14.7	61.7	0.2
w/o custom ISA	31.5	35.8	16.2	61.7	0.6
w/o operator fusion	36.4	42.5	18.9	61.5	1.9
w/o DMA overlap	40.8	47.2	21.4	61.7	3.7
w/o static scratchpad placement	43.6	51.4	22.1	61.6	5.1

The ablation results reveal two important findings. First, memory-centric optimizations dominate overall system behavior. Removing DMA overlap or static scratchpad placement degrades both mean and tail latency more than removing the custom instruction extension. Second, operator fusion has a multiplicative effect because it simultaneously lowers kernel launch overhead, shrinks

intermediate tensor traffic, and improves vector-lane utilization. The custom ISA path still matters, but its impact is concentrated in convolution-heavy layers rather than uniformly across the graph.

Figure 4 summarizes this behavior using a normalized composite score. The score compresses latency, energy, and deadline adherence into a single visualization for qualitative comparison. The largest gap between the full system and ablated variants occurs when DMA overlap is removed, confirming that software-visible dataflow control remains a first-order design parameter.

7 Discussion

The experimental results have three broader implications for real-time AI on RISC-V embedded systems [18].

First, open instruction set architectures are most valuable when paired with disciplined system integration. The custom instruction extension in this study improves performance, but it is not the sole or even primary source of the end-to-end gain. Much of the benefit comes from explicit scratchpad management, bounded runtime behavior, and operator fusion. This suggests that RISC-V's openness should not be interpreted narrowly as an invitation to add hardware features; rather, it enables developers to align software structure, compiler strategy, and architecture around a measurable deployment objective.

Second, the results support a workload-sensitive optimization strategy. Small classification models already perform well after quantization and vectorization, so aggressive hardware specialization offers diminishing returns. By contrast, mid-scale detection models benefit substantially from dataflow-aware scheduling and custom accumulation support. Therefore, edge-system designers should characterize the dominant bottleneck class of their workload before investing in architectural changes. For many products, the best return on effort may come from improving runtime determinism rather than increasing raw arithmetic capability.

Third, real-time suitability requires looking beyond average metrics. The gap between average and p99 latency in the weaker baselines is large enough to produce nontrivial deadline miss ratios. In practical deployments, those misses would manifest as delayed control responses, unstable user interfaces, or dropped sensor events. The proposed system reduces tail latency because it replaces opportunistic buffer management with static schedules and bounded DMA windows. This finding aligns with classical real-time systems principles and reinforces their relevance for modern neural inference pipelines.

There are also limitations [19]. The prototype does not evaluate transformer-based models, whose memory access patterns differ from compact CNNs and temporal convolutions. The custom accelerator focuses on convolutional kernels and may yield smaller gains for attention-dominated models unless paired with matrix-centric instructions. Additionally, although the datasets are representative and the measurement protocol is realistic, they remain controlled benchmarks rather than full product deployments. Future work should integrate online learning constraints, thermal adaptation, and mixed-criticality task sets in which AI inference coexists with safety-certified control logic.

8 Conclusion

This paper presented a complete performance optimization study for RISC-V based embedded systems targeting real-time AI applications. Using keyword spotting, human activity recognition, and compact object detection as representative workloads, we demonstrated that effective acceleration emerges from cross-layer co-design rather than isolated kernel tuning. Quantization and vectorization provide a strong baseline, but the most robust gains come from static scratchpad scheduling, DMA–compute overlap, operator fusion, and deadline-aware runtime configuration. The proposed RISC-V co-designed system reduced end-to-end latency by up to 60.5% relative to an unoptimized baseline and by 38.7% relative to a strong software-only RISC-V deployment, while maintaining accuracy and substantially improving energy efficiency.

The broader conclusion is that RISC-V is a credible platform for real-time edge intelligence when its openness is leveraged to coordinate architecture, compiler, and runtime design. For researchers and practitioners, the main lesson is clear: optimizing embedded AI requires equal attention to numerics, memory movement, and temporal predictability. Future RISC-V edge platforms should therefore expose richer software control over dataflow and scheduling, not merely higher peak compute density.

References

[1] A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual, Volume I: User-Level ISA*, Technical Report UCB/EECS-2014-54, University of California, Berkeley, 2014.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[3] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," in *Proc. IEEE International Conference on Computer Vision (ICCV)*, 2017.

[4] S. Han, H. Mao, and W. J. Dally, "Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding," in *Proc. International Conference on Learning Representations (ICLR)*, 2016.

[5] 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, 2017.

[6] C. L. Liu and J. W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.

[7] K. Asanović et al., "The RISC-V Instruction Set Manual, Volume I: Unprivileged Architecture," RISC-V International, 2021.

[8] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.

[9] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," arXiv:1804.03209, 2018.

[10] D. Anguita, A. Ghoniem, L. Oneto, and X. Parra, "A Public Domain Dataset for Human Activity Recognition Using Smartphones," in *Proc. European Symposium on Artificial Neural Networks*, 2013.

[11] B. Jacob et al., "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference," in *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] T. Chen et al., "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," in *Proc. USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[13] Y. Li, B. Zhang, Z. Zhang, N. Suri, and W. Shi, "A Survey of Edge Intelligence: Architectures, Applications, and Optimization Strategies," *ACM Computing Surveys*, vol. 54, no. 5, pp. 1–36, 2021.

[14] A. González, F. Botta, and J. Abella, "Evaluation of the RISC-V Vector Extension for Embedded AI Workloads," in *Proc. International Conference on Embedded Computer Systems*, 2021.

[15] S. Mittal, "A Survey of FPGA-Based Accelerators for Convolutional Neural Networks," *Neural Computing and Applications*, vol. 32, pp. 1109–1139, 2020.

[16] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[17] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.

[18] N. P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2017.

[19] A. Vaswani et al., "Attention Is All You Need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.