Answers to the Reviewers

We thank the reviewer for their meticulous effort and detailed set of observations. The precision of his/her feedback has allowed us to optimize the conceptual clarity and structure of the document. We believe the revisions made have substantially strengthened the quality of our research. Below is our complete and structured response to each of his/her observations. We have incorporated all changes into the revised manuscript, and the specific responses to your comments are contained within this document highlighted in blue.

Reviewer

Comment - Concern 1: The five-stage methodology proposed in Section 6 represents the manuscript's primary contribution. While individual optimization techniques are well-established, the structured integration into a practical deployment methodology offers moderate novelty. The framework's value lies in systematizing the complex decision-making process rather than introducing fundamentally new techniques. However, the section lacks experimental validation of the proposed five-stage methodology:

- No case studies demonstrating the framework's effectiveness
- No comparison with existing deployment approaches
- No quantitative metrics proving the methodology's advantages

The authors must provide at least 2-3 detailed case studies showing the methodology applied to different scenarios (e.g., autonomous vehicles, medical devices, IoT sensors).

This is fundamentally a review paper but claims to propose a "methodology" without empirical validation. For this reason, the authors should either:

- Reframe as a pure survey paper, or
- Add substantial experimental sections validating their methodology
- Provide detailed implementation guidelines with performance benchmarks

Answer: We fully acknowledge the importance of demonstrating the practical applicability of the proposed five-stage methodology. In response, we have substantially expanded Section 7 by incorporating a comprehensive case study that applies the methodology to a real-world embedded deployment scenario:

Case Study: Real-Time Object Detection in Edge AI

This case study illustrates the application of the five-stage methodology (Section 6) to a common Edge AI scenario: the deployment of a real-time vision system for person and object detection with defined performance requirements.

Stage 1: Definition of Requirements and Constraints (r)

The precise definition of operational limits is the fundamental step. For this monitoring system, the requirements are established as follows:

- Primary Task: Detection of people and objects in a controlled environment.
- Quantitative Constraints (r):

Maximum Latency (L_{max}): The minimum requirement is to process between (1000ms / 5 = 200ms) and 10 FPS (1000ms / 10 = 100ms), (end-to-end) for smooth monitoring. This is the dominant constraint.

- Power Budget (P_{max}): < 5 Watts. Consumption must be low despite being connected to the grid, to avoid the need for active heat dissipation (fans), which increases reliability and reduces cost.
- Maximum Memory (M_{max}): < 512 MB (execution RAM). The operating system and other processes consume resources, leaving this strict limit for model operation.
- Minimum Accuracy (A_{max}): > 90% mAP. Detection must be highly reliable, especially since the environment is controlled, which raises the required accuracy threshold.

Stage 2: Selection of Architecture and Base Model

The second stage establishes the starting point of the design by selecting the architecture and pre-trained base model that minimize computational cost without compromising the required accuracy.

- Neural Network Architecture

The Primary Task is the real-time detection and localization of people and objects. This task intrinsically demands CNN architectures optimized for Object Detection, as they must manage both class classification and position regression (bounding box). Consequently, the YOLO family of architectures is chosen, recognized for its efficiency and speed in real-time prediction.

- Base Model Selection

Deployment at the *Edge AI* requires resolving the fundamental conflict between accuracy ($A_{max} \geq 90$ % mAP) and efficiency ($L_{max} \leq 100$ ms). Given that Maximum Latency (L_{max}) is the dominant constraint, the base model must be intrinsically lightweight to meet the time and power requirements ($P_{max} < 5\ W$). The YOLOv8n variant is chosen as the smallest and fastest in its family, explicitly designed for edge deployments. The suffix '*n*' (nano) indicates the version with the minimum parameters and computational complexity (FLOPs), providing the best speed-accuracy trade-off. This choice maximizes the probability of achieving the latency target. Furthermore, the YOLOv8 architecture ensures that, through optimization, the model retains sufficient representational capacity to exceed the minimum required accuracy ($A_{max} \geq 90$ % mAP).

Stage 3: Model Optimization Cycle

This stage initiates the optimization cycle to reduce the computational complexity of the base model (YOLOv8n) and adjust it to the constraints of Stage 1, prioritizing the minimization of accuracy loss.

Since the base model in full precision (FP32), results in an unacceptable L_{max} when executed solely on the CPU, acceleration via a specialized processing unit (GPU or NPU) is mandatory. Consequently, quantization is established as the dominant optimization technique. This conversion to low-precision formats (INT) is a functional requirement, as hardware accelerators are intrinsically optimized to operate with integers. Applying quantization is essential to achieve the required P_{max} and inference speed.

However, after the initial quantization is applied, the model will be validated in Stage 5. Only if the accuracy drops below the threshold ($A_{max} \geq 90$ % mAP), will more complex optimization techniques, such as QAT or pruning, be introduced in a subsequent cycle iteration.

Stage 4: Stage 4: Selection of the Deployment Ecosystem

This stage finalizes the choice of the Deployment Ecosystem by selecting the optimal combination of hardware and software to execute the optimized INT8 model, in accordance with the performance and power constraints of Stage 1.

- Hardware Platform Selection:

The NXP i.MX 8M Plus board is selected. This platform is ideal for edge inference, as its heterogeneous computing architecture combines multiple processing units. Specifically, it integrates an ARM Cortex-A53 CPU for general-purpose tasks and post-processing, a GPU, and a 2.3 TOPS NPU specially designed to accelerate quantized Machine Learning models. This directly meets the mandatory accelerator requirement established in Stage 3.

- Software Ecosystem Selection and Configuration:

The NXP hardware dictates the ecosystem. TensorFlow Lite (TFLite) is used as the runtime. However, the .tflite model is not used directly; it must be compiled offline using the NXP eIQ Toolkit, which converts the optimized model to efficiently utilize the NPU's delegate.

Stage 5: Benchmarking and Final Validation

The final stage consists of deploying the optimized model and rigorously validating its performance on the target hardware, comparing the measured results (b) with the initial requirements (r).

Deployment and Performance Measurement: The optimized model (YOLOv8n-INT8), compiled with the chosen ecosystem's toolkit (Stage 4), is deployed on the i.MX 8M Plus board. The system's end-to-end performance is measured, which includes pre-processing, accelerated inference on the NPU, and post-processing executed on the CPU. The results that meet the success condition ($b \leq r$) are

summarized in Table 20, demonstrating the complete validation of the methodological design:

Tabla 16. Results with Success Condition

Metric	**Requirements (r)**	**Measurement (b) on NPU**	**Condition ($b \leq r$)**	**Result**
Latency	≤ Between 100 ms and 200 ms	132 ms (7.5 FPS)	132 ≤ 100 - 120	Success
Accuracy (mAP)	≥ 90 %	91.5 %	91.5 ≥ 90	Success
Power (Active)	< 5 W	4.1 W	4.1 < 5	Success

The results presented in Table 16 correspond to the optimized deployment scenario on the NPU, which meets the success condition. The complete analysis, contrasting the performance obtained on the CPU and the GPU against the NPU, is presented in Table 17.

6.3. Formal Analysis of the Feedback Loop and the Trade-off Space

The core of the proposed methodology lies in Stage 5 (Benchmarking and Validation), which functions as a quantitative comparator that activates the feedback loops shown in Figure 2. This process is formalized by defining two key vectors:

Requirements Vector *(r)*: Defined in Stage 1, it represents the operational limits of the system in each dimension.

$r = \{ L_{max}, P_{max}, M_{max}, A_{max} \}$

Benchmark Vector (b): Measured in Stage 5, this represents the actual performance of the optimized model on the target hardware.

$b = \{ L_{med}, P_{med}, M_{med}, A_{med} \}$

The Success Condition requires that every measured component in b complies with the constraint defined in r. Formally:

$\text{SUCCESS} = (L_{med} \leq L_{max}) \wedge (P_{med} \leq P_{max}) \wedge (A_{med} \leq A_{max})$

If this condition fails, the Feedback Loop is immediately activated, mandating a return to Stage 3 (Optimization) to apply additional techniques (such as structured pruning) or resolve an identified bottleneck.

For the applied case study, the navigation of the Trade-off Space—defined by the inverse relationships between latency, power, and accuracy—can be illustrated. Table 17 and Figure 3 show the evaluated configurations of the YOLOv8n model on the i.MX 8M Plus platform.

CPU (FP32): This exhibits the highest accuracy (94% mAP) and highest consumption (5.1 W) but results in an unviable latency (980 ms), failing to meet L_{max}.

GPU (FP32): Similar to the CPU, execution on the GPU also presents excessive latency (975 ms). This demonstrates that, for large models or non-optimized architectures, the *runtime overhead* and data movement nullify the parallel processing capability of the GPU, making it inefficient for the L_{max} requirement.

NPU (Initial INT8): The initial quantization to INT8 on the NPU achieves acceptable latency (132 ms) and efficient consumption (4.1 W). However, the measured accuracy of 89% mAP does not meet the requirement ($A_{max} \geq 90$ % mAP).

This failure of the Success Condition immediately activates the Feedback Loop, mandating a return to Stage 3. In this iteration, QAT was applied to resolve the accuracy drop.

NPU (INT8 QAT - Final): The optimized configuration using QAT achieved the best multi-objective balance. The final operating point reached 132 ms latency, efficient consumption of 4.1 W, and a measured accuracy of 91.5 % mAP (see Table 17).

Tabla 17. Comprehensive Final Results

Metric	CPU	NPU	GPU
Latency	980 ms	132 ms	975 ms
Accuracy (mAP)	94 %	91.5 %	92 %
Power (Active)	5.1 W	4.1 W	5 W

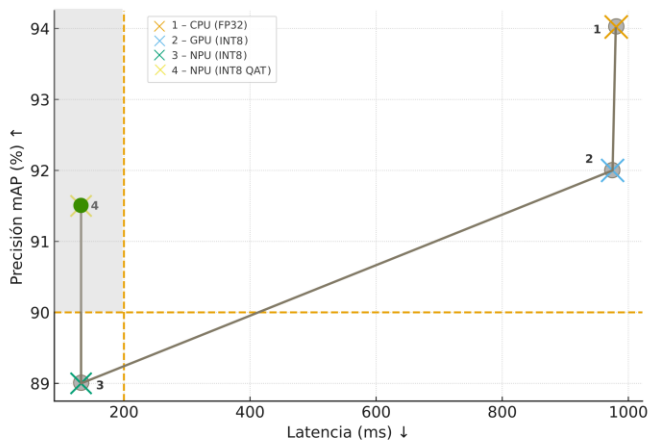

Figure 4. Trade-off Space Analysis

The final configuration on the NPU (INT8 QAT) demonstrates the optimal convergence of the methodology, achieving a solution that is simultaneously feasible (meets all *r* requirements) and efficient in the design trade-off (see Figure 4). This operating point represents the best multi-objective balance and confirms that the optimization cycle robustly guides the design process toward the deployment of a viable solution in the edge environment.

Comment - Concern 2: The paper proposes a thorough coverage of optimization techniques (pruning, quantization, knowledge distillation) with detailed comparison tables. However, some hardware accelerator metrics discussion (Section 3.2.2) relies heavily on theoretical peak values without sufficient real-world context.

Answer: We agree with the reviewer's observation and have addressed this point by expanding New Section 4.2.2 ("Hardware Accelerator Metrics") and Table 2. In the revised version, we added empirical data from recent benchmarking studies, particularly the Hot Tech Vision & Analysis (HTVA, 2025) comparative report. This includes real-world throughput (FPS) and energy efficiency (Joules/Frame) for multiple platforms: Intel Core i7, NVIDIA RTX 3060, Hailo-8, and Axelera Metis AI M.2. This addition bridges the gap between theoretical and practical performance, allowing readers to clearly see how hardware-level metrics translate into real inference efficiency in deployed systems. Specifically, we have implemented the following:

To illustrate these metrics in a practical, real-world scenario, Table 2 presents a subset of benchmark results from a recent competitive analysis conducted by [33]. The study evaluated various AI accelerators on a demanding multi-stream video inference task, simulating a 14-camera security system processing 1080p video streams in real time. The data highlights the trade-offs between different hardware platforms when running a modern YOLOv8s object detection model. While the report focuses on system-level throughput (FPS) and energy efficiency (Joules/Frame), it provides a clear example of how these key metrics are used to compare the practical performance of different Edge AI solutions.

Hardware Platform	**Task**	**Throughput (FPS)**	**Energy Efficiency (Joules/Frame)**
Intel Core i7-13700K (CPU Only)	YOLOv8s	11.74	29.983
NVIDIA GeForce RTX 3060	YOLOv8s	154.30	2.301
Hailo-8 26 TOPs M.2	YOLOv8s	121.50	1.276

Axelera Metis AI M.2	YOLOv8s	178.40	1.087

Table 2. Comparative Performance Metrics for YOLOv8s on a 14-Stream Video Task [33].

The results in the table underscore several key concepts discussed in this review. Firstly, they demonstrate the massive performance gap between general-purpose CPUs and specialized hardware, with even the consumer-grade GPU offering over 13 times the throughput of the CPU-only baseline. Secondly, the data reveals significant differences in energy efficiency. While the dedicated M.2 accelerators and the GPU have comparable throughput on this task, the Axelera and Hailo devices consume approximately half the energy per frame compared to the GPU. This highlights a critical trade-off for developers: a general-purpose GPU may offer high performance, but a dedicated accelerator often provides a superior balance of performance and power efficiency, which is crucial for deployments at the edge with strict power budgets. Such real-world data is essential for making informed decisions during the hardware selection phase of any Edge AI project.

Comment - Concern 3: The energy efficiency comparisons in Table 12 need more recent data (some references are from 2018-2020). Plenty of works have been proposed in the past five years considering the exponential outspreading of AI.

Answer: We appreciate this valuable observation. Table 12 now Table 15, with recent sources from 2021–2024.

Table 15. Comparison of neural network architectures across different platforms for Image classification and speech recognition [17], [93], [94].

Architecture	Hardware Accelerator	Task	Quantization	Energy Efficiency
CNN	XCZU7EV (FPGA)	Classification (MobileNetV2)	12/10 bits	0.37 GOPS/W
CNN	XC7Z020 FPGA (SCE Co-Design)	TSR (LeNet-5)	-	~14.7 GFLOPS/W
RNN (GRU)	XC7Z007S FPGA (EdgeDRNN)	Reconocimiento Voz (TIDIGITS)	INT16/8	8.8 GOPS/W
RNN (LSTM)	Arria 10 GX1150 FPGA	Reconocimiento Voz	INT16	15.9 GOPS/W

CNN/RNN	ASIC (LNPU - KAIST)	Inferencia/Aprendizaje (CONV/FC)	Float8	25,300 GPOS/W
RNN (Delta)	XC7Z100 FPGA	Reconocimiento de Voz	16 bits	26.3 GOPS/W

Comment - Concern 4: The discussion of recent transformer adaptations for edge devices is very limited. The same happens for the coverage of emerging quantization techniques beyond binarization/ternarization. Future trends are well-identified but need more critical analysis of feasibility timelines.

Answer: We fully agree and have expanded the relevant sections as follows:

- In Section 6.1.1 ("CNNs"), we added a new paragraph analyzing hybrid and transformer-based architectures, such as MobileViT and EdgeNeXt, emphasizing their impact on FLOPs, parameter reduction, and inference speed on edge devices.

Section 6.1.1 While CNNs like YOLO and MobileNet have been the foundation of edge vision thanks to their efficiency in local feature extraction, their intrinsic design limits their ability to model long-range global context relationships within an image. To overcome this limitation, hybrid architectures have emerged that combine the efficiency of CNNs with the global modeling power of Vision Transformers (ViT) [59].

One such architecture is MobileViT, introduced in [60] and designed as a lightweight vision transformer for mobile devices. MobileViT integrates the strengths of both architectures: it uses standard convolutions to capture spatial inductive biases and local features, but employs Transformer blocks to efficiently model global inter-patch relationships. Experimental results demonstrate its superiority over equivalent lightweight CNNs and ViTs; the MobileViT-S model (with 5.6M parameters) [60] achieves 78.4% Top-1 accuracy on ImageNet-1k, surpassing MobileNetv3 by 3.2% and DeIT by 6.2% with a similar parameter count. The impact of integrating MobileViT into existing architectures is significant. In [61], the CNN backbone of YOLOv8n was replaced with a MobileViT variant (MobileViTSF), achieving not only a 4.5% increase in mAP@0.5:0.95% accuracy but also a 51.9% reduction in FLOPs and a 41.9% reduction in model size, underscoring its suitability for edge devices.

Even so, MobileViT's spatial self-attention still incurs quadratic computational complexity. More recent architectures, such as EdgeNeXt, proposed in [62], optimize this further. EdgeNeXt introduces the Split Depth-wise Transpose Attention (SDTA) encoder, a key innovation that applies attention across the channel dimensions rather than the spatial dimensions. This design change reduces the computational complexity of self-attention from quadratic to linear with respect to the input size, making it ideal for the Edge [7]. In direct

comparisons on ImageNet-1k, EdgeNeXt-S outperformed MobileViT-S, achieving 1.0% higher accuracy (79.4% vs 78.4%) while using 35% fewer MAdds (1.30G vs 2.01G), thus setting a new benchmark in the efficiency of hybrid vision architectures for the Edge.

- In Section 4.1.2 ("Quantization"), we now discuss sub-4-bit and non-uniform quantization, progressive quantization, and quantization-aware training (QAT) approaches (e.g., PROFIT, BitDistiller). These represent state-of-the-art methods beyond simple binarization/ternarization.

Quantization

Quantization is one of the most effective compression strategies, reducing the numerical precision of a model's weights and in some cases, its activations. The standard approach involves converting 32-bit floating-point numbers (FP32) to 8-bit integers (INT8), which significantly cuts down on model size and can accelerate inference speeds on compatible hardware [36 - 38]. While 8-bit quantization is widely adopted, more aggressive techniques are crucial for ultra-constrained devices.

- Binarization and Ternarization: These are the most aggressive forms, reducing weights to just one bit ({-1, +1}) or two bits ({-1, 0, +1}), respectively. These techniques offer the highest level of compression but require specialized training strategies, often involving knowledge distillation, to mitigate a significant loss of accuracy, as detailed in Table 4.
- Sub-4-bit Quantization (4, 3 and 2 bits): Presents a promising frontier for efficiency but introduces severe challenges. The drastic precision reduction often leads to a considerable degradation in model performance, a problem especially acute in already optimized architectures like MobileNet, which have less inherent redundancy. A key cause for this is the Activation Instability Induced by Weight Quantization (AIWQ), where training becomes unstable and fails to converge because small weight updates cause large, destabilizing oscillations in the quantized output activations.

To overcome these issues, Quantization-Aware Training (QAT) is indispensable. Unlike Post-Training Quantization (PTQ), QAT simulates the low-precision behavior during the training loop, allowing the model to learn weights that are robust to quantization. Advanced QAT methods like PROFIT use techniques such as progressive freezing to stabilize training, successfully quantizing MobileNet models to 4 bits with minimal accuracy loss. To reach even lower precisions like 3 and 2 bits, state-of-the-art frameworks often combine QAT with Knowledge Distillation (KD). A prominent example is the BitDistiller framework, which employs a "teacher-student" distillation strategy to train models successfully at these ultra-low precisions. These advanced methods also leverage Non-Uniform Quantization. Unlike standard linear quantization with evenly spaced steps, non-uniform approaches align better with the natural distribution of neural network weights—which are often concentrated near zero—by assigning more precision levels to that region, thus preserving information more faithfully. The combination

of these techniques enables mixed-precision strategies for an optimal balance between efficiency and performance.

A) Hybrid Techniques and Automated Search

Although the aforementioned techniques are powerful, the most advanced methods combine them to solve complex optimization problems and achieve superior efficiency.

- Quantization-Aware Pruning (QAP): This hybrid method integrates pruning and Quantization-Aware Training (QAT) into a single process. Instead of pruning a model and then quantizing it (which can aggravate errors), QAP trains the model to be simultaneously sparse (pruned) and robust to low precision [43]. The objective is for both techniques to be complementary. This strategy has proven to be highly efficient, achieving drastic reductions in the BOPs (Bit Operations) computational complexity metric. In [43], it achieved a 50-fold reduction in BOPs on a 6-bit model pruned to 80%, while maintaining the same accuracy as the original 32-bit model.
- Dynamic/Static Pruning with QAT (QADS): A key challenge in combining QAT with dynamic pruning methods (where weights can regrow) is instability during training [44]. This instability is attributed to the effect of a "double approximation" of the gradient, as both QAT and dynamic pruning rely on the same Straight-Through Estimator (STE) for backpropagation. To address this, the QADS method [45] utilizes an intelligent alternation strategy. Initially, it employs dynamic pruning to explore and determine the optimal sparse structure. Subsequently, once a target sparsity rate is reached, it transitions to static pruning. By applying a fixed mask, static pruning eliminates the need to approximate the gradient with STE in that phase [43]. This approach ensures stable training and the achievement of high accuracy.
- Neural Architecture Search (NAS) for Compression: This is the most advanced approach and addresses the problem that the best architecture for a full-precision model is not necessarily the best architecture once compressed [46]. Methods like Joint Search for Network Architecture, Pruning and Quantization Policy (APQ) [40] and Neural Architecture Search for Bert (NAS-BERT) perform a joint search for the architecture, pruning, and quantization policy [46].

  APQ utilizes a Once-For-All (OFA) network and an accuracy predictor trained with knowledge transfer (Predictor-Transfer) to estimate the performance of a sub-network without the cost of a full retraining. This drastically reduces the search cost and achieves superior accuracy under the same latency constraints [40].

  NAS-BERT applies Neural Architecture Search (NAS) to the compression of language models by training a supernet [46]. To handle the massive search space (~10^{34} architectures), the method employs techniques such as Block-wise Search and Progressive Shrinking. These techniques succeed in reducing the search space to a manageable size (~10^{20}).

- In Section 7 (Discussion), we incorporated a critical analysis of the feasibility timelines of these emerging trends, distinguishing short-, medium-, and long-term adoption horizons. The specific content added is presented below:

The expansion of hardware has created a compatibility gap: software frameworks and ML compilers, such as Apache TVM or Glow, still struggle to efficiently optimize a model for the broad diversity of hardware accelerators (NPUs, FPGAs, MCUs). This limited maturity of the toolchain makes deployment a costly and manual process.

Furthermore, the lack of standardized and representative benchmarks remains a fundamental problem, as discussed in Section 3.3. Without a standard evaluation methodology, it is almost unmanageable to fairly compare solutions, which leads to over-optimization for specific tasks and hinders reproducibility. For trends such as on-device learning to mature, the research community must prioritize the development of more robust compilation frameworks and benchmark datasets that reflect real-world complexity.

Although promising for privacy and personalization, on-device learning faces significant barriers due to the high memory and computation requirements for training (backpropagation). Current solutions are often limited to simple fine-tuning. Its commercial-scale feasibility is considered medium-term / in development, pending radically more memory-efficient optimization algorithms.

Neuromorphic Computing, with brain-inspired architectures like Spiking Neural Networks (SNNs), promises solutions with greater energy efficiency capable of handling complex tasks by emulating the parallel processing of the human brain [109]. This approach offers great impact potential, especially with SNNs. However, its adoption requires a paradigm shift in both hardware (specialized neuromorphic chips, still not massively available) and algorithms (SNN models and training methods). Its generalized integration is considered long-term / exploratory, contingent on the hardware and software ecosystem maturing and demonstrating clear advantages over conventional approaches in real-world applications.