Performance Analysis of Lightweight Object Detection Networks on On-Device NPU for Security-Critical Industrial Applications

Hyunwoo Kim
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
kim.hw@etri.re.kr

Sung Jae Yoon
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
sj.yoon@etri.re.kr

Munyoung Lee
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
munyounglee@etri.re.kr

Seung Hyub Jeon
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
shjeon00@etri.re.kr

Shin Yuk Kang
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
ameba@etri.re.kr

Kyu Sung Lee
Regional ICT Research Section
ETRI
Daejeon, Republic of Korea
kyusung.lee@etri.re.kr

***Abstract*—AI-based machine vision technologies have been widely applied in various domains, including semiconductor manufacturing, display inspection, autonomous driving, and defense surveillance, to enhance productivity and ensure operational safety. However, these applications often involve sensitive data, necessitating on-device inference rather than cloud-based processing. On-device environments impose strict constraints on computational resources, power consumption, and thermal dissipation, making the adoption of lightweight AI models and low-power Neural Processing Units (NPUs) essential for efficient execution. To provide practical guidelines for model selection and deployment strategies across application domains, this study systematically analyzes the performance of representative lightweight object detection models on an on-device NPU. The analysis includes network architecture examination, inference performance prediction through NPU simulation, and empirical performance measurement on an NPU System-on-Chip (SoC).**

***Index Terms*—On-device NPU, Lightweight Object Detection, Security-Critical, SSD, YOLO.**

I. Introduction

In recent years, artificial intelligence (AI)-based machine vision technologies have played a pivotal role in diverse industrial sectors, enabling advances in quality control, defect detection, and situational awareness, thereby contributing to improved productivity and operational safety. Representative application domains include semiconductor manufacturing processes, display panel inspection, environmental perception for autonomous vehicles, smart city video monitoring, medical image analysis and defense and surveillance systems. These applications often involve data that are sensitive or confidential, where external leakage could result in severe economic, social, or safety-related consequences. Consequently, there is a rapidly growing demand for on-premise and on-device AI inference, rather than transmitting data to the cloud for processing.

On-device inference environments are subject to stringent constraints in terms of computational resources, power consumption, thermal dissipation, and physical footprint. As a result, the development of lightweight AI models and compact, low-power Neural Processing Units (NPUs) capable of executing such models efficiently has become essential. To address these needs, extensive research has been conducted in lightweight model architecture design [1]–[3], computation optimization [4]–[6], and on-device NPU architecture design [7], [8]. However, the requirements for analytical accuracy, processing latency, and power consumption vary across application domains, and the performance characteristics of even the same lightweight model may differ depending on hardware architecture and memory hierarchy design. In security-critical domains, selecting the optimal model–hardware combination that satisfies performance, latency, and power constraints without compromising accuracy becomes essential. This requirement, in turn, necessitates quantitative performance analysis.

This study systematically evaluates the performance of representative lightweight object detection models on an on-device NPU, targeting deployment in security-sensitive industrial environments such as semiconductor manufacturing. The evaluation encompasses network architecture analysis, inference performance prediction through on-device NPU simulation, and empirical performance measurements on an NPU System-on-Chip (SoC). Based on the results, practical guidelines are provided for model selection and deployment

This work was supported by the Technology Innovation Program (Development and Practice of an On-device AI Functionality and Performance Testing Framework based on NPU, RS-2025-02307650, 50%) funded by the Ministry of Trade, Industry & Energy (MOTIE, Korea) and ETRI grant funded by the Korea government (25ZT1100, 25YT1100, 50%).

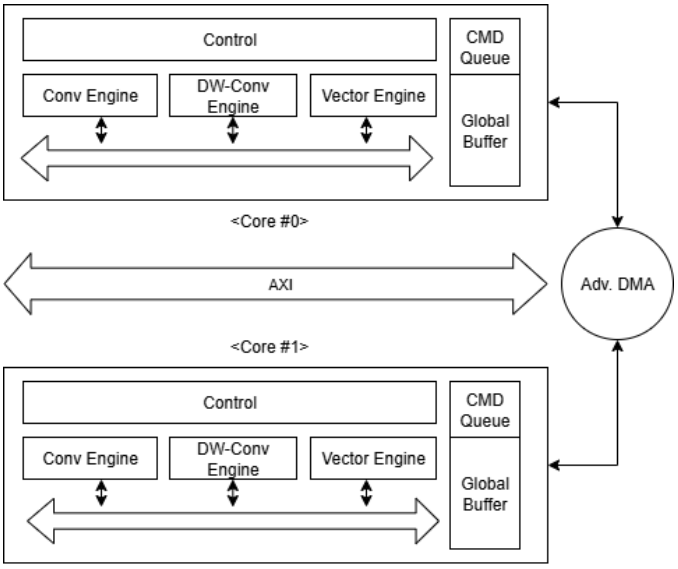


Fig. 1. NPU architecture

strategy formulation tailored to specific application domains.

II. On-device Neural Processing Unit

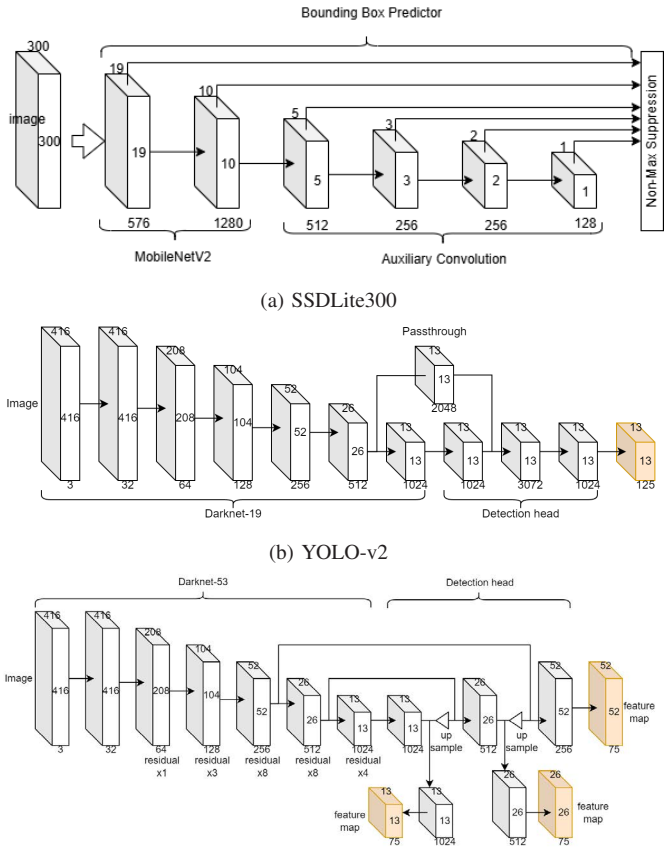
This study employed an NPU SoC designed for on-device inference of convolutional neural network (CNN)-based models to conduct performance analysis. The NPU processes network inference operations using 4-bit or 8-bit integer arithmetic to improve power, performance, and area (PPA) efficiency as well as DRAM bandwidth utilization. Accordingly, model quantization to 8-bit or 4-bit precision is required.

The NPU core comprises a Convolution Engine, a Depthwise Convolution Engine, and a Vector Engine, and includes on-chip buffers for high-speed access to weights, activations, and temporal data. The Convolution Engine handles both standard convolution and pointwise convolution operations, while the Vector Engine performs elementwise addition. The NPU adopts a dual-core architecture and connects to other on-chip IP blocks via an AXI bus.

The SoC integrates a Cortex-A53 ARM core and an Advanced DMA controller for data transfer, interfaced with 2 GB of 32-bit LPDDR4X DRAM. Operating at 1.45 GHz, the NPU delivers 4 TOPS per core, providing a total throughput of 8 TOPS in the dual-core configuration.

III. Lightweight Object Detection Model

The SSDLite [5], YOLO-v2, and YOLO-v3 [9], [10] architectures are employed in this study, as they are widely adopted in lightweight NPU environments owing to their compact model size, low computational complexity, and favorable accuracy–efficiency trade-off. For NPU inference, both activations and weights are quantized to 8-bit integers. All models are trained on the PASCAL VOC 2007 dataset [11]. For SSDLite, two input resolutions are evaluated: SSDLite300 (300×300) and SSDLite512 (512×512). The detection performance,



(a) SSDLite300

(b) YOLO-v2

(c) YOLO-v3

Fig. 2. Lightweight object detection network architectures

measured in terms of mAP, for each model in both FP32 and INT8 precision is presented in Table I.

TABLE I
Accuracy of networks trained on VOC2007 dataset (mAP)

Bit Precision	SSDLite300	SSDLite512	YOLO-v2	YOLO-v3
FP32	0.703	0.744	0.735	0.757
INT8	0.696	0.738	0.734	0.758

The SSDLite architecture utilized in this work is based on the detection network of MobileNetV2 [5], and Fig. 2 (a) illustrates the SSDLite300 model with an input resolution of 300×300. SSDLite512 differs only in the input size and the spatial resolution of each layer, while maintaining the same overall network topology. The SSDLite models adopt MobileNetV2 as the backbone and utilize a total of six feature maps for object detection. The YOLO models employed in this study are YOLO-v2 and YOLO-v3, which were introduced around the same period as SSDLite and exhibit comparable accuracy, thereby serving as baselines for comparison. The architectures of YOLO-v2 and YOLO-v3 are depicted in Fig. 2 (b) and (c), respectively. YOLO-v2 employs the Darknet-19 backbone, resulting in a shallower and simpler structure than SSDLite, whereas YOLO-v3 adopts the Darknet-53 backbone, yielding a deeper and more complex architecture. In addition,

TABLE II
INT8 QUANTIZED NETWORK MODEL SIZES (MB)

Component	SSDLite300	SSDLite512	YOLO-v2	YOLO-v3
Input	0.26	0.75	0.49	0.49
Output	0.21	0.59	0.02	0.25
Activation	13.55	37.18	18.13	54.11
Weight	3.70	3.58	64.03	58.82

TABLE III
OPERATION COUNTS OF NETWORK MODELS (MOPS).
DW = DEPTH-WISE, PW = POINT-WISE

Operation	SSDLite300	SSDLite512	Yolo-v2	Yolo-v3
Conv.	38.88	113.25	16,917.44	34,491.45
DW Conv.	52.11	137.48	–	–
PW Conv.	597.27	1,539.95	553.78	3,095.80
Add.	0.40	1.13	–	15.92
Total	747.82	1,922.78	17,644.27	37,603.17

a simplified version of YOLO-v3 is used in this study, in which the 19th and 20th convolutional layers ($13 \times 13 \times 1024 \times 1024$) of the detection head are removed. Since the detection head accounts for approximately 75% of the total parameters, this modification substantially reduces the model size.

The parameter size and operation count of each model are summarized in Tables II and III, where the operation count accounts only for the weighted layers and the addition operations in the residual structures. The model size of SSDLite is considerably smaller than that of YOLO because it employs depthwise separable convolutions. For activations, SSDLite also requires less memory for the same reason; however, due to its shallower network depth, YOLO-v2 generates fewer activations than SSDLite512. Compared with SSDLite300, SSDLite512 produces a much larger number of activations because of its higher spatial resolution, while its weight size is slightly smaller. Although the same model architecture is used and thus the weight size should ideally be identical, the auxiliary convolution structures introduced to accommodate the difference in spatial resolution make the SSDLite512 architecture slightly more compact, leading to a smaller weight size. In contrast, YOLO-v2 has a substantially larger detection head, resulting in more weights than even the deeper and more complex YOLO-v3.

The operation count increases in the order of SSDLite300, SSDLite512, YOLO-v2, and YOLO-v3, with the YOLO models requiring several times more operations than SSDLite. This is primarily due to the use of depthwise separable convolutions in SSDLite, which substantially reduce the overall number of operations. In addition, it can be observed that pointwise convolutions dominate the operations in SSDLite, whereas standard convolutions account for the majority of operations in the YOLO models.

IV. EXPERIMENTS

A. Experimental Environment

To evaluate NPU performance according to the characteristics of object detection networks, inference was conducted on both an NPU simulator and an NPU SoC. Quantization and compilation of the trained network models, as well as NPU-based inference simulation, were performed using the NPU software toolkit. The NPU simulator estimates inference performance based on the compiled network model and the modeled NPU architecture. The NPU inference performance was measured using the TOPST AI-G board [12], which is an on-device–level single board computer (SBC) operating on a 5 V, 5 A power supply.

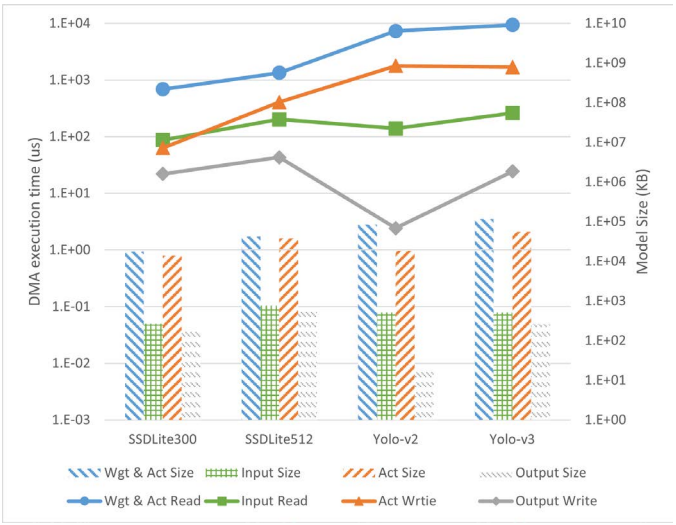


Fig. 3. Model sizes (KB) and DMA transfer time (μs)

B. Data Transfer Time Analysis

To analyze the data transfer latency of the NPU with respect to network model size, the time required for Direct Memory Access (DMA) transfers within the NPU was measured through simulation. Table IV presents the DMA transfer times and data sizes between DRAM and the NPU on-chip buffer, excluding the latency of NPU command transfers. As expected, DMA size and transfer time exhibit a proportional relationship. Since SSDLite models contain substantially fewer weights and activations than YOLO models, their DMA sizes and transfer times are correspondingly several times smaller.

Fig. 3 illustrates the relationship between network model size and DMA transfer time. The bars represent the total size of weights, activations, inputs, and outputs for each model, whereas the lines denote the DMA execution times of the models. For DMA read operations, the DMA transfer time is proportional to each of the weight, activation, and input sizes in the network model. However, although the total weight and activation size of YOLO-v2 is only about twice that of SSDLite512, its DMA read time is approximately 11.5 times longer. This discrepancy can be attributed to the fact that YOLO-v2 has roughly 17.9 times more weights than SSDLite512 while consisting of far fewer layers, resulting in exceptionally large weights per layer. Consequently, the limited capacity of the on-chip memory cannot accommodate the entire weight of each layer, leading to frequent DMA reads.

TABLE IV
DMA EXECUTION TIMES (μS) AND TRANSFER SIZES (KB) OF NETWORK MODELS

Category	Type	SSDLite300	SSDLite512	Yolo-v2	Yolo-v3
DRAM Read	Wgt & Act	688 (5657)	1337 (11496)	7322 (67405)	9385 (86412)
	Input	87 (361)	202 (824)	139 (568)	262 (1066)
	Sum	775 (6017)	1539 (12319)	7461 (67973)	9647 (87479)
DRAM Write	Act	63 (565)	407 (3725)	1772 (16377)	1692 (15701)
	Output	22 (188)	43 (383)	2 (21)	24 (222)
	Sum	85 (752)	450 (4108)	1775 (16398)	1716 (15923)
Total		860 (6770)	1990 (16428)	9235 (84372)	11363 (103402)

TABLE V
EXECUTION TIMES OF COMPUTATION ENGINES (μS)

Engine	Core	SSDLite300	SSDLite512	Yolo-v2	Yolo-v3
Conv.	0	478	977	6621	15946
Engine	1	457	986	6096	15290
DW-C	0	282	698	4	0
Engine	1	261	690	3	0
Vector	0	12	34	11	333
Engine	1	12	33	11	331
Total	0	773	1709	6636	16279
	1	729	1710	6111	15621
	Max	773	1710	6636	16279

For DRAM write operations, output data are written only once at the end of inference; thus, output size and transfer time are directly proportional. In contrast, activation writes exhibit a different pattern. Between SSDLite300 and SSDLite512, the activation size increases by a factor of 2.75, yet the DMA write time grows by about 6.44 times. This can be attributed to the weight-stationary dataflow of the NPU, where limited on-chip buffer capacity significantly increases the DRAM write frequency for activation data. Comparing SSDLite512 with YOLO-v2, the total activation size is halved, but the write time doubles because the large per-layer weights of YOLO-v2 occupy most of the on-chip memory, leaving insufficient space for activations. Finally, in the comparison between YOLO-v2 and YOLO-v3, the activation size triples while the write time increases only slightly, as YOLO-v3 has a much deeper architecture with smaller per-layer weights, allowing more activations to be stored on-chip.

C. Computation Time Analysis

To assess the impact of both inter-model variations and the intra-model distribution of operation types on the NPU's execution time, simulations were performed in which the execution time was measured individually for each operation type. Table V presents the execution time of each NPU computation engine, while Fig. 4 shows the operation count of each model and the corresponding NPU execution time on a logarithmic scale. It can be observed that standard convolutions and pointwise convolutions account for the majority of the operation count, and consequently, the convolution engine responsible for processing these operations dominates the overall execution time. As the operation count of a model increases, the execution time also increases in an approximately linear manner; however, the growth in execution time is

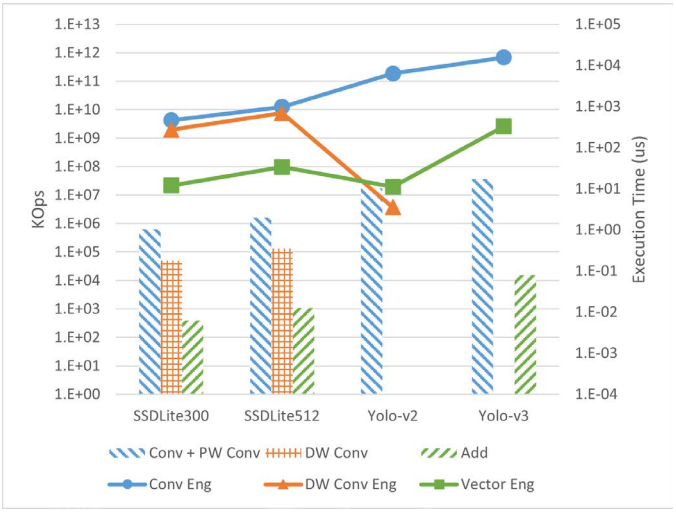


Fig. 4. Execution times (μs) and operation counts (KOps) by operation type

TABLE VI
EXECUTION TIMES (MS) AND NPU UTILIZATION (%) OF NETWORK MODELS

Case	SSDLite300	SSDLite512	Yolo-v2	Yolo-v3
Simulation	1.540	3.463	11.261	24.376
Measured (NPU)	2.000	4.280	12.370	28.490
NPU utilization	21.48	26.37	58.40	58.40

less pronounced than the growth rate of the operation count. This indicates that the resource utilization efficiency of the NPU improves as the model size increases. An exception is YOLO-v3, in which the growth in execution time exceeds the growth in operation count when compared with YOLO-v2. A detailed explanation of this phenomenon is provided in Section IV-D. YOLO-v2, owing to its simplified architecture without separable depthwise convolutions and residual blocks, exhibits an operation count of zero for these components. Consequently, the execution times of the depthwise convolution engine and the vector engine are expected to be zero; however, small non-zero values were observed, as these engines were utilized to process miscellaneous operations.

D. Total Execution Time Analysis

To facilitate a comparison of overall execution time across models, measurements were carried out through simulation as well as on a board incorporating an NPU SoC. Table VI and Fig. 5 present the simulated NPU total execution time, the exe-

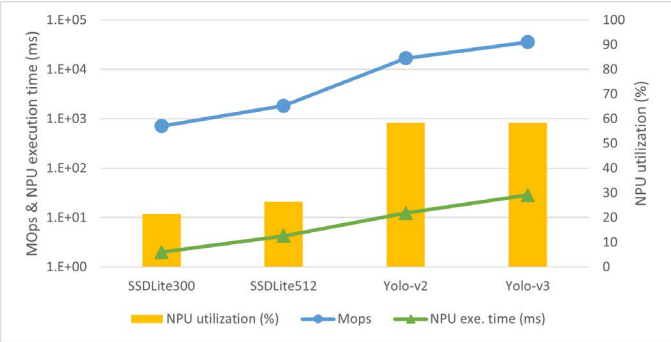


Fig. 5. Relationship among operations count, execution time, and NPU utilization

cution time measured on the NPU SoC, and the corresponding NPU utilization. In the simulation results, the total execution time was observed to be slightly shorter than the simple sum of the DMA transfer time and computation time, because portions of DMA transfers and computations were executed concurrently on separate hardware modules. While the total execution time measured on the NPU SoC deviated slightly from the simulation results, the discrepancy was sufficiently small to validate the reliability of the simulation outcomes.

Overall, as the model size and operation count increased, the total execution time also grew approximately linearly. However, as analyzed in Section IV-C, the rate of increase in the execution time was mitigated by the efficient utilization of NPU resources achieved through dataflow control. When comparing SSDLite512 with YOLO-v2, the operation count differs by a factor of 9.2, whereas the actual execution time differs by only 2.8 times. This discrepancy results from the significant increase in NPU utilization, which rose from 26.37% to 58.4%. Within each model family, the total execution time scaled approximately linearly with the increase in operation count (e.g., SSDLite300 versus SSDLite512, and YOLO-v2 versus YOLO-v3). However, despite the substantial difference in operation count between YOLO-v2 and YOLO-v3, NPU utilization did not improve. This is attributable to the fact that the additional operations primarily resulted from an increased number of layers, which are executed sequentially and therefore do not contribute to improved utilization.

V. Conclusion

This study systematically analyzes the inference performance of representative lightweight object detection models on an on-device NPU, with the aim of providing practical guidelines for model selection and deployment strategies in application-specific scenarios, particularly within security-critical industrial environments such as semiconductor manufacturing. Both SSDLite and YOLO models were evaluated using NPU simulation and on-chip inference on the NPU SoC hardware (i.e., the TOPST AI-G platform), accompanied by a quantitative analysis of the correlations among network complexity, DMA transfer load, computational engine utilization, and overall inference latency. Experimental results demonstrated that, although larger model sizes and operation counts increased the computational workload and execution time, the utilization of NPU computation engines improved correspondingly, thereby moderating the growth rate of latency and enhancing overall computational efficiency. Furthermore, scaling up the model size led to substantial variations in the DRAM access volume for parameters and activations, primarily determined by the NPU's on-chip buffer capacity constraints and dataflow policy. The differences between simulation and hardware measurements remained within an acceptable margin, while the performance trends with respect to model size and operation count were consistent, confirming the validity of the simulation results. The findings of this work provide a practical basis for the design of model architectures and deployment strategies that balance accuracy, latency, and energy efficiency in security-sensitive industrial applications.

References

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[2] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[3] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2704–2713, 2018.

[4] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[5] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

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

[8] N. P. Jouppi, D. H. Yoon, M. Ashcraft, M. Gottscho, T. B. Jablin, G. Kurian, J. Laudon, S. Li, P. Ma, X. Ma, *et al.*, "Ten lessons from three generations shaped google's tpuv4i: Industrial product," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–14, IEEE, 2021.

[9] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[10] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[11] M. Everingham, L. V. Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010. The VOC2007 dataset is described as part of the Pascal VOC Challenge.

[12] TOPST, "Ai-g: Edge ai-focused single board computer." https://topst.ai/product/g/ai, 2025.