Performance Analysis of Deep Learning Model Quantization on NPU for Real-Time Automatic License Plate Recognition Implementation

Daniel Alexander [1]*, Wildanil Ghozi [2]*
* Faculty of Computer Science, Informatics Engineering, Dian Nuswantoro University, Semarang, Indonesia
111202113308@mhs.dinus.ac.id [1], wildanil.ghozi@dsn.dinus.ac.id [2]

Article Info

Article history:

Received 2025-06-02
Revised 2025-06-26
Accepted 2025-07-03

Keyword:

Deep Learning,
Edge Computing,
License Plate Recognition,
Neural Processing Unit,
Quantization.

ABSTRACT

Neural Processing Units (NPUs) are dedicated accelerators designed to perform efficient deep learning inference on edge devices with limited computational and power resources. In real-time applications such as automated parking systems, accurate and low-latency license plate recognition is critical. This study evaluates the effectiveness of quantization techniques, specifically Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), in improving the performance of YOLOv8-based license plate detection models deployed on an Intel NPU integrated within the Core Ultra 7 155H processor. Three model configurations are compared: a full-precision float32 model, a PTQ model, and a QAT model. All models are converted to OpenVINO's Intermediate Representation (IR) and benchmarked using the benchmark_app tool. Results show that PTQ and QAT significantly enhance inference efficiency. QAT achieves up to 39.9% improvement in throughput and 28.6% reduction in latency compared to the non-quantized model, while maintaining higher detection accuracy. Both quantized models also reduce model size by nearly 50 percent. Although PTQ is simpler to implement, QAT offers a better balance between accuracy and speed, making it more suitable for deployment in edge scenarios with real-time constraints. These findings highlight QAT as an optimal strategy for efficient and accurate license plate recognition on NPU-based edge platforms.

I. Introduction

The rapid advancement of Artificial Intelligence (AI) applications has driven a growing demand for efficient inference capabilities on edge environments [1], which refer to resource-constrained devices such as smart cameras, IoT devices, and mobile platforms. Edge computing enables data processing closer to the source, thereby reducing latency, enhancing privacy, and supporting real-time decision-making [2].

To meet these requirements, many semiconductor manufacturers have started incorporating AI accelerators into System-on-Chip (SoC) architectures, with Neural Processing Units (NPUs) emerging as a key component designed specifically to accelerate AI workloads [3]. A prominent example is the Intel® NPU, which offers efficient AI inference while maintaining low power consumption [4].

In automated parking systems, embedded devices typically face significant computational limitations, making it difficult to deploy deep learning models effectively [5]. NPUs offer a practical solution to this challenge, providing a power-efficient yet high-performance platform for real-time and accurate license plate recognition in edge-based parking systems [6].

Deep learning models such as YOLOv8, widely adopted for object detection tasks [7][8], can be applied to license plate recognition in automated parking systems. However, these models are generally trained using 32-bit floating point (float32) representations, which are not optimized for deployment on NPUs. Instead, NPUs, including Intel's, perform more efficiently with 8-bit integer (INT8) representations [4].

To bridge this gap, quantization techniques have emerged as an effective strategy to reduce model complexity by converting float32 representations into INT8 [9]. One such

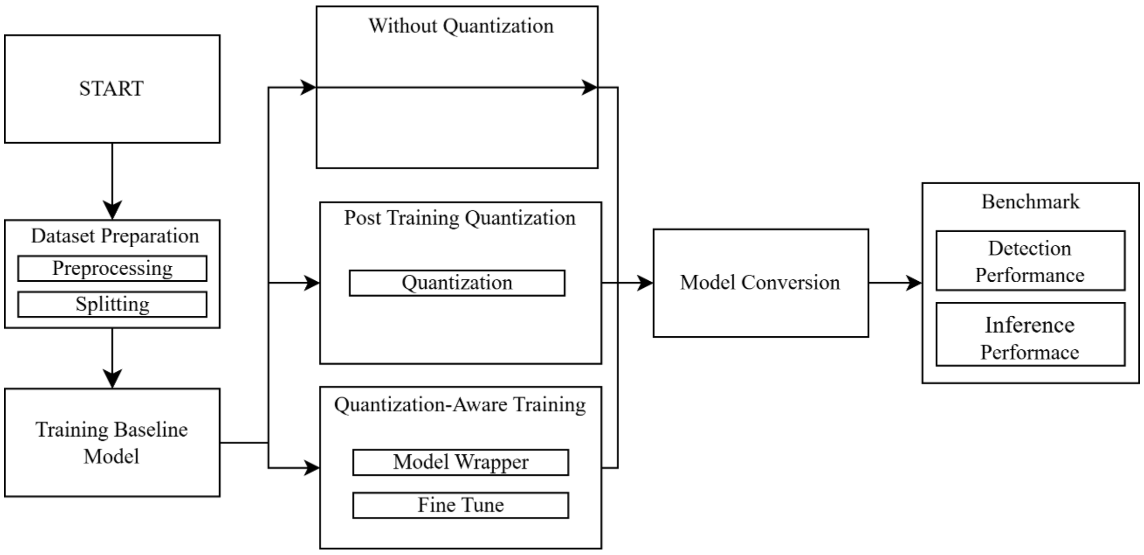

Figure 1 Research stage

approach is quantization-aware training (QAT) [10], which differs from post-training quantization (PTQ) that applies quantization after model training is complete [11][12]. QAT introduces quantization effects during the training phase itself [13][14], allowing the model to adapt its weights and activations accordingly for better INT8 performance [15]. This results in more stable and accurate models when deployed on edge devices like the Intel NPU, which are constrained in both power and computational resources [16].

Deploying deep learning models on edge devices remains a significant challenge due to limited computational resources, power constraints, and real-time processing requirements. In a domain-specific example, Hasiao et al. [17] proposed an FPGA-based License Plate Alignment and Recognition (LPAR) system for edge deployment. While the system reached 95% recognition accuracy using lightweight models, it could only operate at 2 FPS demonstrating the persistent performance bottlenecks in real-time applications under constrained hardware. To overcome these limitations, recent research has explored the use of AI accelerators such as Neural Processing Units (NPUs). Rakshith Jayanth et al. [2] showed that NPUs outperform CPUs by 58.6% in matrix-vector operations and by 3.2× in neural network inference, while consuming less power. Similarly, Fei and Abdelfattah [4] reported a 1.8× speedup when running large language models (LLMs) on NPUs compared to CPUs. These findings support NPUs as a more energy-efficient alternative to GPUs, particularly for edge AI workloads. However, NPUs are not without challenges. Tan and Cao [3] highlighted architectural limitations of NPUs on mobile platforms and stressed the need for optimized frameworks to unlock their full potential. Complementary to this, Krishnamoorthi [18] introduced quantization-aware training (QAT) as an effective approach to maintain high accuracy under low-bit quantization (e.g., 4-bit), enabling efficient deployment of models on resource-constrained edge hardware.

This study addresses the challenge of deploying accurate and efficient Automatic License Plate Recognition (ALPR) systems on edge devices with limited computational resources and power. While YOLOv8 provides strong detection performance, its high computational demands limit real-time use on such platforms. To improve efficiency, this study applies Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT) to compress the model and accelerate inference. The evaluation focuses on detection accuracy and inference speed using the Intel NPU integrated in the Intel Core Ultra 7 155H processor. The three model variants, namely baseline float32, PTQ, and QAT, are benchmarked using a real-world license plate dataset. Instead of merely verifying real-time capability at 30 FPS or faster, the study aims to identify the best trade-off between speed and accuracy. It is important to note that the scope is limited to license plate detection only, excluding optical character recognition and full ALPR integration. The findings offer practical insights into the effectiveness of quantization methods for real-time edge deployment.

II. METHOD

The research process involves several key stages, beginning with data preparation, followed by baseline model training, application of quantization techniques, and evaluation of model accuracy and performance. These stages are integrated into a license plate detection pipeline that is optimized for deployment on edge devices. All experiments are conducted on a system equipped with an Intel Core Ultra 7 155H processor featuring an integrated Neural Processing Unit (NPU), 16 GB of LPDDR5 RAM, and Windows 11 operating system. The complete workflow of the proposed method is illustrated in Figure 1.

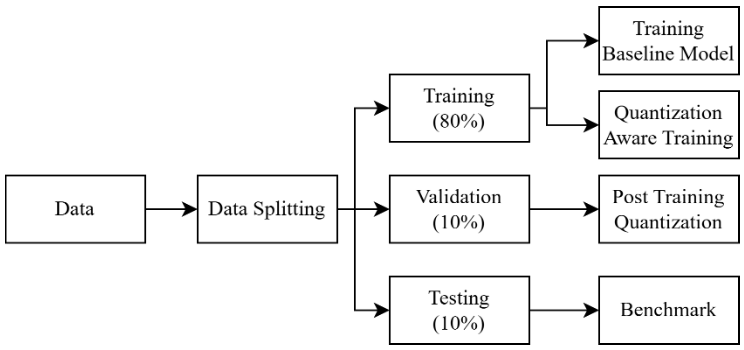

Figure 2. Data flow

A. *Data Preparation*

This study utilizes the Car Plate Detection dataset obtained from the Kaggle platform, comprising 433 annotated images. Each image features a vehicle with a visible license plate, and is accompanied by Pascal VOC-style XML annotations indicating the bounding box of the plate. The dataset contains license plates from various countries, presenting diverse formats, fonts, and backgrounds, thereby supporting model robustness across international ALPR scenarios.

1) *Data Preprocessing*: Annotation files in XML format are parsed and converted into YOLO-compatible text format, where each line represents a normalized bounding box (center x, center y, width, height) and class ID. All images are resized to 320×320 pixels, maintaining aspect ratio where applicable. Since the dataset is relatively small, no synthetic augmentation is applied in this stage to preserve the original image distribution and plate styles.

2) *Data Splitting:* Data splitting is a crucial step to ensure objective and unbiased model evaluation. By separating training and testing data, the model's ability to generalize to unseen data can be properly assessed while minimizing the risk of overfitting [19]. In this study, the dataset is proportionally divided into three subsets: 80% is allocated for training and fine-tuning during the quantization-aware training phase, 10% for validation and calibration during post-training quantization, and the remaining 10% for testing during performance benchmarking. The splitting is performed randomly to ensure that each sample has an equal probability of being assigned to any subset, thereby preventing bias that may arise from manual or uneven partitioning. The training subset is used to build the model, while the validation data monitors its performance throughout the training process. The entire data preprocessing and splitting workflow is illustrated in Figure 2.

B. *Baseline Model Training*

Baseline model was trained using the pre-trained YOLOv8m architecture, which includes pre-initialized weights and activation parameters that can be further fine-tuned on domain-specific datasets. YOLOv8m was chosen for its balance between detection speed and accuracy, making it particularly suitable for real-time applications such as automated parking systems that require efficient vehicle license plate recognition [20]. This model has been widely adopted in various object detection scenarios and provides a solid foundation for further optimization.

The training process was conducted on a custom dataset specifically prepared for license plate detection. The model was trained over 50 epochs to allow it to learn relevant visual patterns and variations across license plate instances. The resulting YOLOv8m model serves as a baseline for performance comparison before applying any quantization techniques. This uncompressed model is used as a reference point to evaluate the trade-offs introduced by model compression methods in later stages.

C. *Post-Training Quantization*

Post-Training Quantization (PTQ) is a quantization technique applied after a model has been fully trained, eliminating the need for retraining [16]. This method converts the model's weights and activations from 32-bit floating point representations to lower-precision formats such as 8-bit integers. By reducing model size and computation complexity, PTQ significantly accelerates inference performance, particularly on edge devices with limited resources [9].

The PTQ process involves a calibration phase using a subset of the validation data (calibration dataset) to determine the minimum and maximum real values (r_{min} and r_{max}) that will be mapped into the quantized integer range. This mapping transforms real values r into quantized values q using an affine transformation formula (Equation 1)

$$q = \frac{r}{s} + z \qquad (1)$$

Where s is the scale factor (a positive real number) and z is the zero-point, which has the same data type as q. There are two main quantization approaches: symmetric and asymmetric.

1) Symmetric Quantization: In symmetric quantization, the zero-point z is fixed at zero, and only the scale factor s is optimized during calibration [21]. The range of the floating-

point representation is defined in Equation 2.

$$[r_{min}, r_{max}] = [-s \cdot q_{max}, s \cdot q_{min}] \quad (2)$$

Quantization is carried out in three stages: scaling, clamping, and rounding (Equation 3).

$$q = [clamp\left(\frac{r}{s}, q_{min}, q_{max}\right)] \quad (3)$$

2) *Asymmetric Quantization:* Asymmetric quantization explicitly optimizes both the lower and upper bounds of the floating-point domain (Equation 4) [21].

$$s = \frac{r_{max} - r_{min}}{2^b - 1}; z = -\frac{r_{min}}{s} \quad (4)$$

D. Quantization-Aware Training

Quantization-Aware Training (QAT) is an advanced optimization technique aimed at preserving high model accuracy after quantization [14]. QAT simulates quantization effects during training, enabling the model to adapt to numerical errors or distortions introduced by low-precision formats such as INT8 [21].

1) *Model Wrapper*: QAT begins by wrapping the baseline model into a new structure that inserts special operations known as FakeQuantization during the forward pass. These operators simulate both quantization and dequantization in a differentiable way, allowing the gradient to flow during backpropagation [21]. FakeQuantization is applied to both weights and activations at each layer using the same affine transformation as in PTQ. By incorporating FakeQuantization early in training, the model becomes “aware” of quantization effects and adjusts its parameters accordingly. This leads to improved robustness when the model is later converted to a low-bit representation, helping to minimize accuracy degradation.

2) *Fine Tuning:* Following the wrapping process, the model undergoes fine-tuning for 10 epochs using the training dataset. This step adjusts model parameters to better accommodate quantization-induced noise without requiring full retraining from scratch. Fine-tuning optimizes existing weights to retain semantic representation and high classification performance despite limited numerical precision. It also aligns internal statistics, such as those from Batch Normalization layers to ensure they remain consistent during post-quantization inference [21].

E. Model Conversion

To enable deployment on the Intel NPU integrated within the Intel Core Ultra 7 155H, all model variants whether non-quantized (baseline model), post-training quantized (PTQ), or quantization-aware trained (QAT) must be converted into OpenVINO’s Intermediate Representation (IR) format. This process begins with the YOLOv8 model exported from PyTorch to ONNX, followed by conversion to IR format using the Model Optimizer (MO) tool from the OpenVINO 2025.2 toolkit. Specific optimization flags are applied to ensure compatibility with intel-npu execution targets, such as enabling INT8 inference and optimizing layer fusion. This conversion ensures that the model can fully utilize the hardware acceleration capabilities of the integrated NPU during runtime.

F. Benchmarking

The benchmarking process in this study focuses on evaluating two primary aspects of model performance: detection accuracy and inference performance. This evaluation is conducted in a controlled environment using synthetic benchmarking tools, without incorporating full-system variables such as camera latency, NPU initialization time, or integration with external ALPR pipelines.

1) *Detection Accuracy:* Model accuracy is assessed using widely accepted object detection metrics: precision, recall, and mean average precision (mAP). Precision reflects how many of the detected license plates are correct, while recall indicates how many actual plates were successfully detected. The mAP is reported at two levels:

- mAP@0.5 measures accuracy at an IoU threshold of 0.5
- mAP@0.5:0.95 measures averages accuracy across multiple IoU thresholds for a more comprehensive evaluation.

These metrics are obtained from the YOLOv8 training logs and are applied to a held-out test set. All three model variants, namely the baseline model, the Post-Training Quantization (PTQ) model, and the Quantization-Aware Training (QAT) model, are evaluated to determine the impact of quantization on detection accuracy.

2) *Inference Performance:* Inference benchmarking is carried out using OpenVINO’s benchmark_app utility, which performs synthetic performance testing on models in Intermediate Representation (IR) format. The models are executed on the Intel NPU integrated in the Core Ultra 7 155H processor. Two key performance indicators are measured:

- Throughput, in frames per second (FPS)
- Latency, in milliseconds (ms) per frame.

All tests are conducted with -d NPU flag to ensures execution on the Intel NPU, with input resolution fixed at 320×320 pixels to match training settings.

This benchmarking evaluates whether each model (float32, PTQ, and QAT) meets the real-time threshold of 30 FPS or 33 ms latency, and identifies which offers the best trade-off between speed and detection accuracy. The focus is not only on meeting real-time requirements, but also on assessing efficiency and performance retention after quantization.

It is important to note that this evaluation is synthetic and does not include system-level factors such as hardware boot time, camera latency, or backend integration. Thus, the results reflect only the core inference performance, not full end-to-end system behavior.

III. RESULT AND DISCUSSION

The evaluation focuses on three approaches for license plate detection using YOLOv8m: a baseline model without quantization, a model optimized through Post-Training Quantization (PTQ), and a model refined using Quantization-Aware Training (QAT). The evaluation includes both detection accuracy metrics and system performance metrics to assess the impact of quantization on model effectiveness and efficiency.

The dataset used was the Car Plate Detection dataset from Kaggle, consisting of 433 annotated images. After preprocessing to format the bounding box coordinates and class labels, the data was split into 345 training images (80%), 44 validation images (10%), and 44 testing images (10%).

The baseline model was trained for 50 epochs using YOLOv8m, and the performance detection evaluation was conducted using metrics including precision, recall, mAP@0.5, mAP@0.5:0.95, average inference latency, throughput (FPS), and model size in megabytes.

A. Baseline Model / Non-Quantization

TABLE I
WITHOUT QUANTIZATION PERFORMANCE MATRIX

Precision	0.941
Recall	0.886
mAP@0.5	0.921
mAP@0.5:0.95	0.549
Avg. Latency (ms)	31.10
Throughput (FPS)	128.35
Model Size (MB)	50.46

The baseline model without quantization achieved high performance across most evaluation metrics. It reached a precision of 0.941 and a recall of 0.886, with a strong mAP@0.5 of 0.921, as shown in Table 1. However, the model size was relatively large at 50.47 MB, which could be a limitation for deployment on edge devices with restricted storage and computational resources.

B. Post-Training Quantization

Post-Training Quantization (PTQ) was applied after training, using INT8 representation without retraining. Calibration was performed using 44 validation images. This method led to some trade-offs in accuracy but significantly improved performance metrics.

The quantitative results of PTQ are summarized in Table 2. Compared to the baseline, PTQ caused a slight drop in all accuracy metrics: precision declined to 0.906 and recall to 0.878, while mAP@0.5 dropped to 0.901. Nonetheless, the model size was reduced by nearly 50% to 25.31 MB. Additionally, average latency decreased by 27.9% and throughput increased by 38.6%, demonstrating PTQ's effectiveness in boosting inference efficiency with minimal accuracy loss.

TABLE II
POST-TRAINING QUANTIZATION PERFORMANCE MATRIX

Precision	0.906
Recall	0.878
mAP@0.5	0.901
mAP@0.5:0.95	0.482
Avg. Latency (ms)	22.43
Throughput (FPS)	177.94
Model Size (MB)	25.31

C. Quantization Aware Training

Quantization-Aware Training (QAT) was implemented using the Neural Network Compression Framework (NNCF). The baseline model was quantized and then fine-tuned for 10 epochs using the training dataset.

TABLE III
QUANTIZATION-AWARE TRAINING PERFORMANCE MATRIX

Precision	0.958
Recall	0.841
mAP@0.5	0.906
mAP@0.5:0.95	0.560
Avg. Latency (ms)	22.22
Throughput (FPS)	179.59
Model Size (MB)	25.37

QAT led to the highest precision among all models at 0.958, indicating increased selectivity in predictions. While recall slightly dropped to 0.841, the mAP@0.5:0.95 metric improved to 0.560, outperforming both the PTQ and the non-quantized model suggesting better generalization across IoU thresholds. Like PTQ, the model size was halved compared to the baseline, and QAT also achieved the best system performance with the lowest latency (22.22 ms) and highest throughput (179.59 FPS), as summarized in Table 3.

Table 4 summarizes the accuracy and performance of the three evaluated models. The full-precision model achieves the highest recall (0.886) and mAP@0.5 (0.921), but with the highest latency (31.10 ms), lowest throughput (128.35 FPS), and largest model size (50.46 MB). The PTQ model significantly reduces latency and model size by nearly 50%, with only a slight drop in accuracy (mAP@0.5: 0.901). The QAT model offers the best overall trade-off, achieving the highest precision (0.958), best mAP@0.5:0.95 (0.560), lowest latency (22.22 ms), and highest throughput (179.59 FPS), while maintaining a compact size of 25.37 MB. These results confirm that QAT delivers both high accuracy and efficiency, making it ideal for real-time edge deployment.

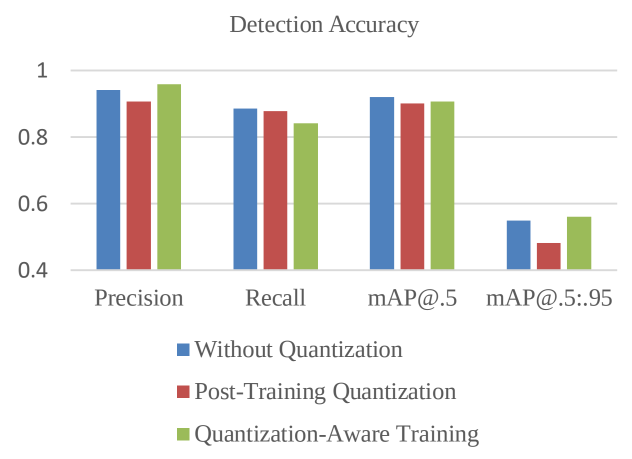

Figure 3. Detection accuracy chart

Figure 3 illustrates the detection accuracy metrics of all three models, clearly showing the impact of quantization.

Figure 4 presents the throughput improvements obtained through model quantization, showing a significant increase in frame processing rate from 128.35 FPS in the baseline model (Without Quantization) to 177.94 FPS with Post-Training Quantization (PTQ) and further to 179.59 FPS with Quantization-Aware Training (QAT). This represents an increase of up to 39.9%, demonstrating how quantization techniques effectively accelerate inference speed. The substantial boost in throughput makes both PTQ and QAT models highly suitable for real-time applications, particularly on edge devices where processing speed is critical.

TABLE IV
DETECTION ACCURACY & INFERENCE PERFORMANCE SUMMARY

Method	**Precision**	**Recall**	**mAP@0.5**	**mAP@0.5:0.95**	**Avg. Latency (ms)**	**Throughput (FPS)**	**Model Size (MB)**
Non-Quantized	0.941	0.886	0.921	0.549	31.10	128.35	50.46
Post-Training Quantization	0.906	0.878	0.901	0.482	22.43	177.94	25.31
Quantization-Aware Training	0.958	0.841	0.906	0.560	22.22	179.59	25.37

Although both PTQ and QAT exhibit some degradation in accuracy compared to the baseline, QAT significantly reduces the performance gap through fine-tuning. This process enables the quantized model to regain much of the lost accuracy and perform closely to the original FP32 model.

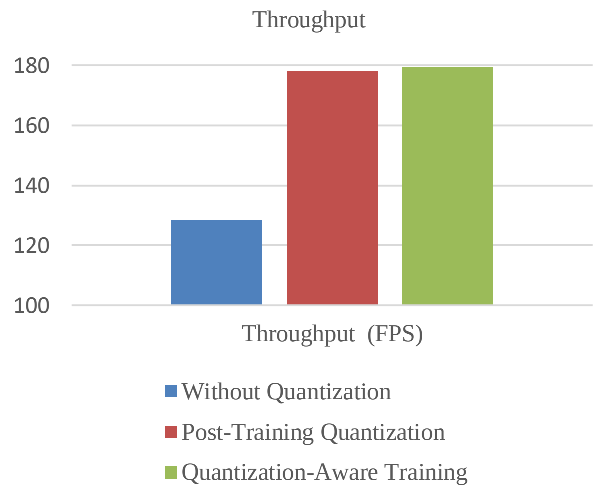

Figure 4. Throughput chart

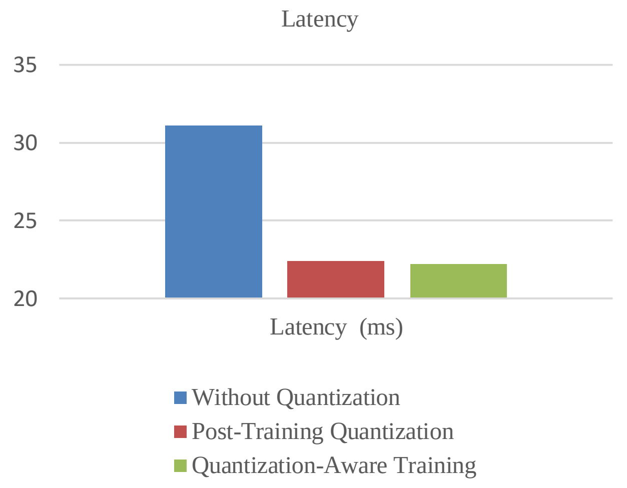

Figure 5. Latency chart

Figure 5 displays the inference latency across all models, highlighting a clear reduction in average latency achieved through quantization. The full-precision baseline model records a latency of 31.10 ms, while Post-Training Quantization (PTQ) reduces it significantly to 22.43 ms, and Quantization-Aware Training (QAT) further lowers it to 22.22 ms. This reduction of nearly 9 milliseconds confirms that quantization not only enhances throughput and reduces model size but also substantially improves inference speed. Such latency improvements are critical for real-time applications, especially on edge devices where responsiveness and efficiency are paramount.

IV. Conclusion

This study demonstrates the effectiveness of quantization techniques, specifically Post-Training Quantization (PTQ) and Quantization-Aware Training (QAT), in improving the deployability of deep learning-based license plate detection systems on edge AI hardware. By targeting deployment on the Intel NPU integrated within the Intel Core Ultra 7 155H processor, the results confirm that quantization not only reduces computational load and model size but also enables real-time inference performance that is suitable for edge environments.

Although the baseline float32 model achieves the highest detection accuracy, it requires significantly higher latency and computational resources. In contrast, both PTQ and QAT offer considerable improvements. PTQ increases throughput by 38.6%, reduces latency by 27.9%, and compresses the model size by nearly 50%. QAT further enhances these gains by achieving a 39.9% increase in throughput, 28.6% reduction in latency, and slightly better model compression, while retaining more of the original model's accuracy. In the benchmarking results, the baseline model reached 128 frames per second, whereas both PTQ and QAT achieved up to 179 frames per second on the Intel NPU.

While PTQ is easier to implement since it does not require retraining, it is more prone to accuracy degradation, especially in datasets with high variability. QAT, which integrates quantization effects during training, proves to be more robust in preserving accuracy under diverse real-world conditions.

In conclusion, QAT is identified as the most suitable approach for optimizing object detection models in edge computing scenarios where real-time performance, resource constraints, and detection reliability must be carefully balanced. Future work may include expanding the dataset, testing under varied lighting and motion conditions, and integrating a complete ALPR pipeline to evaluate full end-to-end system performance on edge NPU platforms.

Bibliography

[1] D. Xu *et al.*, “Edge Intelligence: Empowering Intelligence to the Edge of Network,” *Proceedings of the IEEE*, vol. 109, no. 11, pp. 1778–1837, Nov. 2021, doi: 10.1109/JPROC.2021.3119950.

[2] R. Jayanth, N. Gupta, and V. Prasanna, “Benchmarking Edge AI Platforms for High-Performance ML Inference,” Sep. 23, 2024, *arXiv*: arXiv:2409.14803. doi: 10.48550/arXiv.2409.14803.

[3] T. Tan and G. Cao, “FastVA: Deep Learning Video Analytics Through Edge Processing and NPU in Mobile,” in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Jul. 2020, pp. 1947–1956. doi: 10.1109/INFOCOM41043.2020.9155476.

[4] A. Fei and M. S. Abdelfattah, “NITRO: LLM Inference on Intel Laptop NPUs,” Dec. 15, 2024, *arXiv*: arXiv:2412.11053. doi: 10.48550/arXiv.2412.11053.

[5] R. Al-batat, A. Angelopoulou, S. Premkumar, J. Hemanth, and E. Kapetanios, “An End-to-End Automated License Plate Recognition System Using YOLO Based Vehicle and License Plate Detection with Vehicle Classification,” *Sensors*, vol. 22, no. 23, 2022, doi: 10.3390/s22239477.

[6] R. Laroca *et al.*, “A Robust Real-Time Automatic License Plate Recognition Based on the YOLO Detector,” in *2018 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2018, pp. 1–10. doi: 10.1109/IJCNN.2018.8489629.

[7] M. Hussain, “YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection,” *Machines*, vol. 11, no. 7, 2023, doi: 10.3390/machines11070677.

[8] M. Hussain, “YOLOv1 to v8: Unveiling Each Variant–A Comprehensive Review of YOLO,” *IEEE Access*, vol. 12, pp. 42816–42833, 2024, doi: 10.1109/ACCESS.2024.3378568.

[9] Z. Liu, Y. Wang, K. Han, S. Ma, and W. Gao, “Post-Training Quantization for Vision Transformer,” Jun. 27, 2021, *arXiv*: arXiv:2106.14156. doi: 10.48550/arXiv.2106.14156.

[10] M. Shen *et al.*, “Once Quantization-Aware Training: High Performance Extremely Low-bit Architecture Search,” Sep. 28, 2021, *arXiv*: arXiv:2010.04354. doi: 10.48550/arXiv.2010.04354.

[11] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, “Accurate Post Training Quantization With Small Calibration Sets,” in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., in Proceedings of Machine Learning Research, vol. 139. PMLR, Jul. 2021, pp. 4466–4475. [Online]. Available: https://proceedings.mlr.press/v139/hubara21a.html

[12] Y. Mishchenko *et al.*, “Low-Bit Quantization and Quantization-Aware Training for Small-Footprint Keyword Spotting,” in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Dec. 2019, pp. 706–711. doi: 10.1109/ICMLA.2019.00127.

[13] S. A. Tailor, J. Fernandez-Marques, and N. D. Lane, “Degree-Quant: Quantization-Aware Training for Graph Neural Networks,” Mar. 15, 2021, *arXiv*: arXiv:2008.05000. doi: 10.48550/arXiv.2008.05000.

[14] M. Chen *et al.*, “EfficientQAT: Efficient Quantization-Aware Training for Large Language Models,” Oct. 02, 2024, *arXiv*: arXiv:2407.11062. doi: 10.48550/arXiv.2407.11062.

[15] Z. Liu *et al.*, “LLM-QAT: Data-Free Quantization Aware Training for Large Language Models,” May 29, 2023, *arXiv*: arXiv:2305.17888. doi: 10.48550/arXiv.2305.17888.

[16] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A Survey of Quantization Methods for Efficient Neural Network Inference,” in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022.

[17] C.-H. Hsiao, H. Lee, Y.-T. Wang, and M.-J. Hsu, “Efficient License Plate Alignment and Recognition Using FPGA-Based Edge Computing,” *Electronics*, vol. 14, no. 12, Art. no. 12, Jan. 2025, doi: 10.3390/electronics14122475.

[18] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” Jun. 21, 2018, *arXiv*: arXiv:1806.08342. doi: 10.48550/arXiv.1806.08342.

[19] A. Nurhopipah and U. Hasanah, “Dataset splitting techniques comparison for face classification on CCTV images,” *IJCCS*, vol. 14, no. 4, p. 341, Oct. 2020, doi: 10.22146/ijccs.58092.

[20] M. L. Ali and Z. Zhang, “The YOLO Framework: A Comprehensive Review of Evolution, Applications, and Benchmarks in Object Detection,” *Computers*, vol. 13, no. 12, 2024, doi: 10.3390/computers13120336.

[21] A. Kozlov, I. Lazarevich, V. Shamporov, N. Lyalyushkin, and Y. Gorbachev, “Neural Network Compression Framework for Fast Model Inference,” in *Intelligent Computing*, K. Arai, Ed., Cham: Springer International Publishing, 2021, pp. 213–232. doi: 10.1007/978-3-030-80129-8_17.