Applied Artificial Intelligence Research

INNOVIAIR

A Real-time LPR Deployment Scheme on Edge Devices via INT8 Quantization and TensorRT Optimization

Xi Chen[a,*], Jiajia Chen[a]

[a]School of Intelligent Manufacturing, Chongqing Jianzhu College, Chongqing 400072,China

ARTICLEINFO

Keywords:
License Plate Recognition (LPR)
YOLOV5
Edge Computing
INT8 Quantization

ABSTRACT

Real-time license plate recognition (LPR) on edge devices is heavily constrained by hardware limits. This study focuses on the efficient deployment of a real-time Chinese license plate recognition system, which plays a crucial role in intelligent traffic management and automated toll collection. Despite substantial progress in artificial intelligence and deep learning algorithms, achieving real-time performance remains a challenge because of the limited computational resources of practical hardware platforms. To overcome this limitation, a two-stage model optimization approach is proposed and integrated with NVIDIA deployment toolkits to accelerate inference with only marginal accuracy loss. The proposed work delivers a high-performance license plate recognition system and demonstrates that hardware-aware model optimization is essential for achieving efficient and practical real-time deployment.

1.Introduction

License plate detection and recognition (LPD/LPR) is a fundamental perception task in intelligent transportation systems. It supports traffic monitoring, parking management, automated toll collection, vehicle access control, and urban surveillance[1,2]. In these applications, recognition results must be generated with low latency and sufficient accuracy because delayed or incorrect plate information may affect traffic flow, billing reliability, and security management.

Although deep learning has significantly improved license plate detection and recognition, practical deployment remains constrained by real-world imaging conditions and hardware resources. In outdoor traffic scenes, license plates may appear under illumination variation, motion blur, occlusion, perspective distortion, and complex backgrounds[3,4]. At the same time, many practical systems are expected to run on edge-side devices rather than cloud servers, which imposes strict constraints on computing power, memory bandwidth, inference latency, and video-stream processing capability[5,6].

This study is motivated by the gap between high-accuracy LPR algorithms and real-time edge deployment requirements. A model that performs well in offline testing may still fail to satisfy real-time deployment requirements if preprocessing, inference, post-processing, video decoding, and streaming overhead are not jointly optimized. Therefore, deployment-oriented optimization should consider not only the neural network architecture but also quantization strategy, GPU acceleration, TensorRT engine conversion, and streaming-pipeline integration[7,8].

To address these challenges, this paper proposes an integrated deployment framework for real-time Chinese license plate detection and recognition. The framework combines YOLOv5s for license plate detection and LPRNet for character recognition. To improve deployment efficiency, post-training INT8 quantization, selective mixed-precision retention, TensorRT engine optimization, CUDA-based preprocessing, and DeepStream-based video analytics are incorporated into a unified pipeline.

The main contributions of this study are summarized as follows. First, an end-to-end two-stage LPR deployment framework is developed by integrating YOLOv5s-based detection and LPRNet-based recognition for Chinese license plate scenarios. Second, a selective INT8 quantization strategy is adopted, in which quantization-sensitive detection output layers are retained in FP32 precision to reduce accuracy degradation. Third, CUDA preprocessing, TensorRT inference optimization, and DeepStream pipeline deployment are jointly implemented to improve end-to-end processing efficiency. Fourth, experiments are conducted on both high-performance GPU and edge-device platforms to analyze the trade-off among accuracy, latency, FPS, and hardware constraints.

* Corresponding author.
E-mail address: jlinghu@163.com.
https://doi.org/10.65455/nygma907
Received 28 May 2026; Received in revised form 2 July 2026; Accepted 8 July 2026; Available 20 July 2026

2.Related work

2.1.License plate detection and recognition methods

Early LPR systems mainly relied on handcrafted image features, traditional image processing, character segmentation, and optical character recognition. These methods were effective in controlled scenarios but were sensitive to illumination variation, complex backgrounds, motion blur, and irregular plate layouts. With the development of deep convolutional neural networks, data-driven visual representation has become the dominant technical route for both license plate detection and recognition[3,4].

For detection, single-stage detectors represented by the YOLO family have been widely adopted because they formulate object localization and classification as a unified regression problem[9,10]. Compared with two-stage detectors, YOLO-based methods usually provide a better balance between accuracy and inference speed, making them suitable for real-time traffic applications. YOLOv5s is particularly attractive for edge deployment because of its compact architecture and relatively low computational cost[11,12].

For recognition, segmentation-free sequence prediction models have gradually replaced traditional character segmentation pipelines. Li and Shen formulated license plate recognition as a sequence labeling problem and demonstrated the effectiveness of CNN-LSTM-based recognition without explicit character segmentation[13]. LPRNet is a representative lightweight recognition network that directly predicts license plate character sequences from cropped plate images and uses sequence decoding to obtain the final plate number[14]. This design reduces the error accumulation caused by explicit character segmentation and improves robustness in low-quality plate images.

2.2.Datasets and data augmentation for Chinese LPR

Dataset quality directly affects the robustness of deep learning-based LPR systems. The Chinese City Parking Dataset (CCPD) provides diverse real-world vehicle images with different viewpoints, illumination conditions, weather conditions, and plate positions[15]. It has therefore become an important benchmark for Chinese license plate detection and recognition. However, real-world datasets may still be insufficient for covering rare plate styles and degraded imaging conditions. Synthetic license plate generation can supplement real data by controlling fonts, colors, geometric distortion, blur, noise, and illumination changes, thereby improving recognition robustness[16].

2.3.Quantization and deployment acceleration

Model compression and quantization are important techniques for deploying deep neural networks on edge devices. Post-training quantization (PTQ) converts floating-point weights and activations into low-precision integer representations after training, while quantization-aware training (QAT) simulates quantization effects during training to reduce performance degradation[17,18]. INT8 quantization can reduce memory consumption and accelerate inference, but it may introduce numerical errors that affect layers sensitive to localization regression and confidence prediction.

In addition to model-level compression, deployment frameworks also play an important role in real-time performance. TensorRT accelerates inference on NVIDIA GPUs through layer fusion, kernel auto-tuning, precision calibration, and memory reuse[19]. CUDA can further reduce preprocessing latency by parallelizing pixel-level operations such as resizing and format conversion[20]. DeepStream provides an integrated video analytics pipeline that combines video decoding, preprocessing, inference, post-processing, and visualization, which is essential for practical streaming deployment[21].

2.4.Research gap

Existing studies have achieved substantial progress in LPR algorithms, model compression, and GPU-accelerated inference. However, many works focus on improving detection or recognition accuracy in isolation, while fewer studies evaluate the complete detection-recognition-deployment pipeline under practical streaming conditions. As a result, the relationship between model-level optimization and end-to-end system performance remains insufficiently analyzed.

Moreover, the layer-wise sensitivity of INT8 quantization in two-stage LPR systems has not been fully investigated. Directly applying global INT8 quantization may reduce latency but can damage localization accuracy if quantization-sensitive output layers are not properly handled. The interaction among quantization, TensorRT optimization, CUDA preprocessing, and DeepStream deployment also requires further analysis, especially across heterogeneous hardware platforms such as desktop GPUs and low-power edge devices.

Therefore, this study focuses on the deployment-oriented optimization of a Chinese LPR system. Different from studies that only report model accuracy or single-model inference speed, this work evaluates detection accuracy, recognition accuracy, quantization sensitivity, preprocessing latency, TensorRT inference latency, and end-to-end FPS under different hardware configurations.

3.Methodology

3.1.Overall framework

The proposed framework adopts a two-stage detection-recognition architecture, as shown in Fig 1. The input video frame is first decoded and preprocessed. A YOLOv5s detector then localizes license plate regions. The detected regions are cropped and resized before being sent to the LPRNet recognition model. Finally, the decoded character sequence is overlaid on the original video stream together with the bounding box and confidence score.

The deployment pipeline is designed according to the logical order of video input, preprocessing, license plate detection, region cropping, character recognition, post-processing, and streaming output. Model-level optimization is

performed through INT8 quantization and TensorRT engine conversion, while system-level acceleration is achieved through CUDA-based preprocessing and DeepStream pipeline integration. This design allows the influence of each module on accuracy and latency to be analyzed separately.

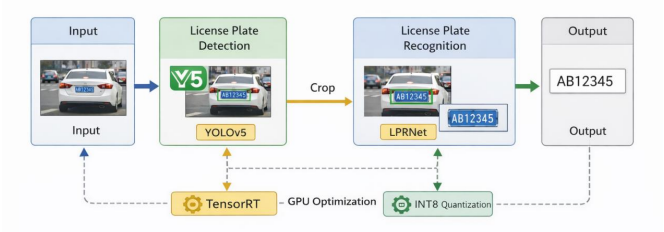

Fig 1. Overall framework of the proposed real-time LPR deployment pipeline

3.2.Dataset preparation

Two datasets were used for model training and validation. For license plate detection, 10,000 images were selected from the Chinese City Parking Dataset (CCPD), including 8,000 images for training and 2,000 images for validation. The annotation information embedded in the CCPD filenames was converted into YOLO format. Each image was resized to 640 × 640 pixels and normalized before being fed into the detector.

Detection data augmentation included random rotation, brightness adjustment, scaling, and geometric perturbation. Unless otherwise specified, the augmentation range was set to small-angle rotation, random brightness variation, and multi-scale resizing to simulate common traffic-surveillance conditions. These operations were used to reduce overfitting and improve robustness under illumination variation, tilted plates, and different shooting distances.

For license plate recognition, a synthetic Chinese license plate dataset containing 100,000 images was generated. The dataset was divided into 80,000 training images and 20,000 validation images. The synthetic generation process considered province abbreviations, letters, digits, plate colors, font variations, perspective distortion, motion blur, Gaussian noise, and brightness changes. The recognition images were cropped or resized to the input resolution required by LPRNet before training.

To avoid ambiguity, all training and validation subsets were kept independent. For the synthetic recognition dataset, the character sequences in the validation set were generated independently from those in the training set. This setting was used to reduce the risk of duplicated plate strings between training and validation samples.

3.3.License plate detection using YOLOv5s

YOLOv5s was selected as the detector because it provides a favorable balance between detection accuracy and computational efficiency. The model consists of a CSPDarknet backbone, a PANet-based feature fusion neck, and a multi-scale detection head. The backbone extracts hierarchical visual features, the neck fuses semantic and spatial information across resolutions, and the detection head predicts bounding box coordinates, objectness scores, and class confidence values.

The detector was initialized with pretrained weights and fine-tuned on the CCPD detection subset. The input size was set to 640 × 640 pixels, and the model was trained for 50 epochs. Adam was used as the optimizer, and early stopping was applied when the validation performance no longer improved. During inference, non-maximum suppression (NMS) was used to remove duplicate bounding boxes. Detection performance was evaluated using precision, recall, mAP@0.5, and mAP@0.5:0.95.

3.4.License plate recognition using LPRNet

After license plate localization, each detected plate region is cropped from the original frame and forwarded to LPRNet. LPRNet is a lightweight segmentation-free recognition model. Instead of detecting and segmenting individual characters, it directly predicts a character sequence from the entire plate image, thereby reducing error propagation caused by inaccurate character segmentation.

The recognition network extracts visual features through convolutional layers and converts them into a sequence representation. The output sequence is decoded using Connectionist Temporal Classification (CTC), which allows the model to handle variable-length character sequences without requiring frame-level character alignment. The model was trained on the synthetic Chinese license plate dataset for 10 epochs. Plate-level exact-match accuracy was used as the main validation indicator, while character-level accuracy and normalized edit distance were added as complementary metrics in the revised evaluation protocol.

3.5.Model optimization and deployment

3.5.1.INT8 quantization and calibration

To reduce memory consumption and inference latency, post-training INT8 quantization was applied during TensorRT engine construction. A floating-point value x is mapped to an integer value q according to the scale factor s and zero point z:

$$q = \mathrm{clip}(\mathrm{round}(\frac{x}{s})+z, q_{min}, q_{max}) \quad (1)$$

For signed INT8 quantization, q_{min} = -128 and q_{max} = 127. The corresponding dequantization process is defined as:

$$\hat{x} = s \cdot (q - z) \quad (2)$$

Where $\hat{x}$ denotes the reconstructed floating-point approximation. For symmetric quantization, z is usually set to 0 and the scale factor can be estimated as $s = \frac{\max(|x|)}{127}$. For asymmetric quantization, $s = (x_{max} - x_{min}) / (q_{max} - q_{min})$ and $z = \mathrm{round}(q_{min} - \frac{x_{min}}{s})$.

Calibration was performed using representative samples selected from the validation distribution. These samples were used to estimate activation ranges and determine quantization parameters. Layer-wise sensitivity analysis was then conducted by comparing the detection accuracy before and after quantizing different modules. The detection output layer was found to be sensitive to INT8 numerical errors because it directly predicts bounding box coordinates and confidence scores. Therefore, this layer was retained in FP32 precision, while convolutional and addition operations were quantized to INT8.

3.5.2.TensorRT engine optimization

After training and calibration, the models were converted into TensorRT engines. TensorRT improves inference efficiency through graph-level and kernel-level optimizations, including layer fusion, kernel auto-tuning, precision calibration, and memory reuse. Layer fusion reduces intermediate memory access between adjacent operators. Kernel auto-tuning selects efficient GPU kernels according to the target hardware. Memory reuse reduces repeated memory allocation and improves runtime efficiency.

The deployment toolchain included PyTorch for model training, PyTorch-quantization for INT8 calibration, TensorRT for engine construction, CUDA for preprocessing acceleration, and DeepStream for video-stream pipeline integration. The same preprocessing and post-processing logic was maintained during both validation and deployment to ensure consistent inference behavior.

3.5.3.CUDA-based preprocessing acceleration

Image resizing and format conversion may become bottlenecks in real-time video analytics. To reduce preprocessing latency, bilinear interpolation was implemented on the GPU using CUDA. For an output pixel at location (u, v), the corresponding source coordinate is mapped to (x, y). The pixel value is computed using the weighted average of the four nearest source pixels:

$$I(u,v)=(1-\alpha)(1-\beta)I(x_0,y_0)+\alpha(1-\beta)I(x_1,y_0)+(1-\alpha)\beta I(x_0,y_1)+\alpha\beta I(x_1,y_1) \quad (3)$$

where $\alpha = x - x_0$ and $\beta = y - y_0$. In the CUDA implementation, different output pixels are assigned to different GPU threads, allowing thousands of interpolation operations to be executed in parallel.

3.5.4.DeepStream streaming deployment

The optimized TensorRT engines were integrated into a DeepStream pipeline. The pipeline consisted of video input decoding, frame batching, CUDA preprocessing, YOLOv5s detection, plate-region cropping, LPRNet recognition, post-processing, visualization, and output streaming. YOLOv5s was configured as the primary inference module, while LPRNet was used as the secondary recognition module for detected plate regions.

Deployment performance was evaluated on both an RTX 3090 GPU and a Jetson Nano edge device. The comparison was used to analyze whether model-level acceleration can be translated into end-to-end system-level speedup under different hardware constraints.

3.6.Evaluation metrics

The proposed system was evaluated from two aspects: recognition effectiveness and deployment efficiency. For license plate detection, the Intersection over Union (IoU) between the predicted bounding box B_p and ground-truth bounding box B_g is defined as:

$$IoU=\frac{Area(B_P \cap B_g)}{Area(B_P \cup B_g)} \quad (4)$$

Precision and recall are defined as follows:

$$Precision = \frac{TP}{TP+FP} \quad (5)$$

$$Recall = \frac{TP}{TP+FN} \quad (6)$$

The mean Average Precision at IoU = 0.5 is reported as mAP@0.5, while mAP@0.5:0.95 averages AP values over IoU thresholds from 0.5 to 0.95 with a step of 0.05. The latter is stricter and more sensitive to localization quality.

4.Results and discussion

To empirically validate the proposed two-stage framework, we conducted a series of baseline evaluations and ablation studies. The core objective was to dissect the tangible impacts of INT8 quantization, CUDA-accelerated preprocessing, and heterogeneous hardware constraints on the end-to-end inference pipeline.

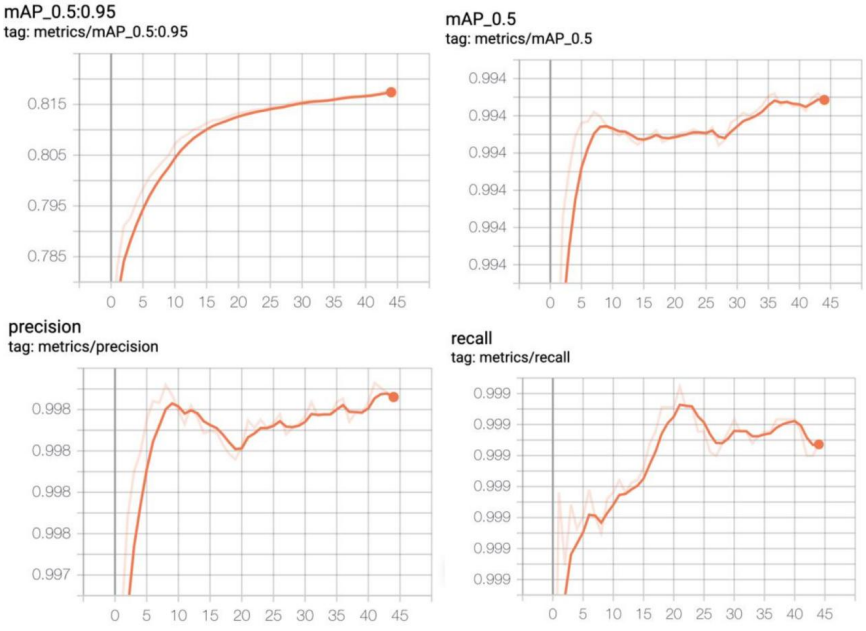

Fig 2. Training curves of mAP@0.5:0.95, mAP@0.5, precision, and recall for the LPD model

4.1.Experimental setup and baseline performance

To ensure a reliable and reproducible evaluation environment, the initial model training and baseline benchmarking were executed on an NVIDIA RTX 3090 GPU. The software stack relied heavily on standardized NVIDIA Docker containers, specifically leveraging PyTorch for LPD training, the TAO Toolkit for LPRNet fine-tuning, TensorRT for engine optimization, and DeepStream for pipeline integration.

The LPD network was trained using a split of 8,000 training images and 2,000 validation images. Over a 50-epoch training cycle, the baseline FP32 model demonstrated exceptional robustness. It achieved a training mAP of 99.41% at an IoU threshold of 0.5, alongside a precision of 99.82% and a recall of 99.89%. Furthermore, the model maintained high fidelity under the more stringent 0.5:0.95 IoU threshold, recording an 83.9% mAP on the validation set. This minimal divergence between training and validation metrics indicates that the detection network successfully avoided overfitting while preserving strong generalization capabilities across challenging scenarios.

Table 1. Training and validation results of the LPD model

	mAP_0.5	mAP_0.5:0.95	Precision	Recall
Training Result	99.41%	81.77%	99.82%	99.89%
Validation Result	99.40%	83.90%	99.70%	99.80%

The strong detection performance can be attributed to the multi-scale feature extraction capability of YOLOv5s. License plates often occupy only a small portion of the image and may appear under varying scales and viewing angles. The PANet-based feature aggregation mechanism enables effective fusion of semantic and spatial information across different resolutions, thereby improving the detection of small targets. Furthermore, transfer learning using pretrained weights accelerates convergence and enhances feature representation, contributing to the high precision and recall observed on both training and validation sets.

Similarly, the LPR model, tasked with character deciphering, was trained on a substantially larger dataset of 80,000 images and validated against 20,000 images. After merely 10 epochs, the network converged to a training accuracy of 99.87% and a marginally higher validation accuracy of 99.88%. This consistency highlights the network's high reliability in translating cropped bounding boxes into accurate text arrays.

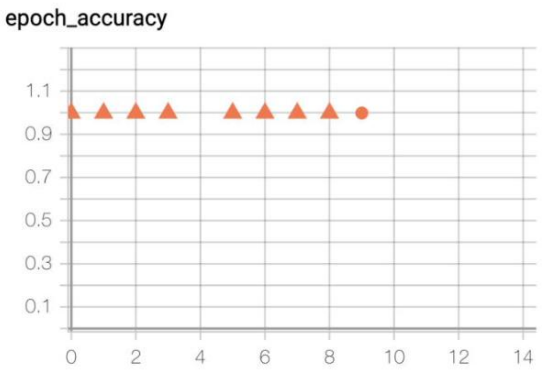

Fig 3. Training accuracy curve of the LPR model

Table 2. Performance of the LPR model

	Accuracy
Training Result	99.87%
Validation Result	99.88%

The high recognition accuracy achieved by LPRNet is mainly due to its end-to-end sequence prediction architecture. Unlike traditional OCR-based methods that require explicit character segmentation, LPRNet directly predicts character sequences from the entire license plate image. This design avoids segmentation errors and improves robustness under conditions such as motion blur, low resolution, and partial occlusion. In addition, the use of synthetic license plate images increases the diversity of training samples, thereby enhancing generalization capability.

4.2.Impact of INT8 quantization and layer sensitivity

Post-training INT8 quantization was first applied to the LPD model to evaluate the trade-off between numerical precision and deployment efficiency. The results show that global quantization of Conv2d and Add modules reduced mAP@0.5:0.95 from 83.90% to 80.20%, while mAP@0.5 remained nearly unchanged. This indicates that coarse localization ability was preserved, but fine-grained bounding box quality was affected by quantization noise.

The main reason is that the detection output layer of YOLOv5s directly predicts bounding box offsets, objectness scores, and classification confidence values. Even small INT8 rounding errors may be amplified during anchor-based decoding and NMS. The stricter mAP@0.5:0.95 metric is more sensitive to such localization deviations than mAP@0.5. Therefore, retaining the detection output layer in FP32 precision can reduce quantization-induced localization errors while still allowing most convolutional computation to benefit from INT8 acceleration.

The layer-wise sensitivity experiment confirmed this interpretation. When the detection output layer was excluded from INT8 quantization, mAP@0.5:0.95 recovered from 80.20% to 83.60%, which is close to the FP32 baseline of 83.90%. This selective mixed-precision strategy therefore provides a practical compromise between accuracy preservation and inference acceleration.

Table 3. Original LPD model vs. global PTQ vs. selective PTQ with FP32 detection output layer

	mAP_0.5	mAP_0.5:0.95	Precision	Recall
Original LPD Model	99.40%	83.90%	99.70%	99.80%
PTQ(Conv2d+Add)	99.40%	80.20%	99.60%	99.80%
PTQ+Disable Detection Layer Quantization	99.40%	83.60%	99.70%	99.80%

The selective quantization strategy also improved deployment efficiency. After TensorRT conversion, the average inference latency decreased from 1.53 ms to 0.71 ms. The warm-up time decreased from 46.33 ms to 10.59 ms, indicating that TensorRT engine optimization and INT8 execution can substantially reduce runtime overhead. These results suggest that quantization should not be applied

uniformly to all layers; instead, quantization-sensitive layers should be identified and protected according to their functional roles in the network.

Table 4. Warm-up and inference latency before and after selective INT8 quantization

	Warm-up	Inference
Original LPD Model	46.33ms	1.53ms
PTQ+Disable Detection Layer Quantization	10.59ms	0.71ms

4.3.CUDA-Accelerated Preprocessing

Image preprocessing, especially resizing and cropping, is a frequent bottleneck in CPU-bound video analytics pipelines. In the proposed framework, a CUDA-based bilinear interpolation kernel was implemented to move preprocessing from the CPU to the GPU.

The preprocessing benchmark showed that PIL required 14.154 ms, OpenCV required 1.248 ms, and the CUDA implementation required 0.026 ms for the tested preprocessing operation. Although OpenCV is optimized for CPU execution, it still relies on limited CPU-level parallelism compared with the massive thread-level parallelism available on the GPU.

The speedup is mainly explained by the computational structure of image resizing. Bilinear interpolation computes each output pixel independently from neighboring source pixels. This operation is naturally parallelizable: CUDA assigns different output pixels to different GPU threads, so thousands of pixel computations can be executed concurrently. In addition, keeping preprocessing on the GPU reduces synchronization overhead between preprocessing and TensorRT inference. Therefore, CUDA preprocessing helps prevent the inference engine from waiting for input frames.

Table 5. Preprocessing latency comparison among PIL, OpenCV, and CUDA

	Preprocessing Time
PIL	14.154ms
OpenCV	1.248ms
CUDA	0.026ms

It should be noted that preprocessing latency may vary depending on image resolution, batch size, memory-copy strategy, and whether host-to-device transfer time is included. Therefore, the CUDA result should be interpreted as evidence of the potential advantage of GPU-side preprocessing rather than as an isolated replacement for complete end-to-end pipeline evaluation.

4.4.End-to-End pipeline evaluation and hardware constraints

The optimized models and preprocessing kernels were integrated into a DeepStream pipeline for end-to-end deployment evaluation. The pipeline completed video decoding, detection, plate cropping, recognition, post-processing, and visualization. The output stream displayed bounding boxes, recognized Chinese license plate characters, confidence scores, and the number of detected plates.

Fig 4. DeepStream output of the proposed LPR pipeline on a parking-lot scene

On the RTX 3090 platform, the pipeline achieved approximately 30 FPS, which satisfies the common real-time video threshold of 25 FPS. This result indicates that the optimized detector, recognizer, CUDA preprocessing, and TensorRT engines can support real-time LPR processing on a high-performance GPU platform.

When the same pipeline was deployed on Jetson Nano, the throughput decreased to approximately 4-5 FPS. This result shows that model-level INT8 quantization and TensorRT optimization alone are insufficient to guarantee real-time performance on low-tier edge devices. The performance gap is related not only to neural network inference but also to hardware-level constraints, including GPU computing capability, memory bandwidth, video decoding throughput, CPU-GPU data transfer, and DeepStream scheduling overhead.

This finding provides an important deployment insight: for practical edge-side LPR systems, acceleration should be evaluated at the pipeline level rather than only at the single-model inference level. On devices with limited resources, further optimization may require a lighter detector, reduced input resolution, frame-skipping strategy, asynchronous decoding, batch-size tuning, or deployment on more capable edge platforms such as Jetson Xavier or Jetson Orin.

5.Conclusion

This study presented a real-time Chinese license plate recognition deployment framework integrating YOLOv5s detection, LPRNet recognition, selective INT8 quantization, TensorRT optimization, CUDA preprocessing, and DeepStream-based streaming deployment. The revised framework emphasizes both model accuracy and system-level deployment efficiency.

Experimental results show that the YOLOv5s detector achieved 99.40% mAP@0.5 and 83.90% mAP@0.5:0.95 on the validation set, while the LPRNet model achieved 99.88% validation accuracy on the synthetic recognition dataset. For deployment optimization, preserving the detection output layer in FP32 precision reduced the quantization-induced mAP@0.5:0.95 degradation from 3.70 percentage points to 0.30 percentage points. Meanwhile, TensorRT-based INT8 deployment reduced inference latency from 1.53 ms to 0.71 ms.

The end-to-end DeepStream pipeline achieved approximately 30 FPS on the RTX 3090 platform, demonstrating real-time processing capability. However, the same pipeline achieved only 4-5 FPS on Jetson Nano, indicating that low-tier edge deployment is constrained not only by model inference but also by memory bandwidth, video decoding, CPU-GPU data transfer, and streaming-pipeline overhead.

Future work will focus on quantization-aware training, more lightweight detection backbones, real-world cross-scene testing, character-level recognition error analysis, and deployment on more capable edge devices. In addition, more detailed pipeline profiling will be conducted to distinguish the latency contributions of decoding, preprocessing, inference, post-processing, and visualization.

References

[1] LAROCA R, SEVERO E, ZANLORENSI L A, et al. A robust real-time automatic license plate recognition based on the YOLO detector//International Joint Conference on Neural Networks. Rio de Janeiro, Brazil: IEEE, 2018: 1-10.

[2] MASOOD S Z, SHU G, DEHGHAN A, et al. License plate detection and recognition using deeply learned convolutional neural networks [Z/OL]. arXiv:1703.07330, 2017. https://arxiv.org/abs/1703.07330.

[3] LI H, WANG P, SHEN C. Towards end-to-end car license plate detection and recognition with deep neural networks. IEEE Transactions on Intelligent Transportation Systems, 2019, 20(3): 1126-1136.

[4] SELMI Z, HALIMA M B, PAL U, et al. DELP-DAR system for license plate detection and recognition. Pattern Recognition Letters, 2020, 129: 213-223.

[5] MINH H T, MAI L, MINH T V. Performance evaluation of deep learning models on embedded platform for edge AI-based real-time applications//8th NAFOSTED Conference on Information and Computer Science. Hanoi, Vietnam: IEEE, 2021: 350-355.

[6] BUI T Q V, NGUYEN V T, KIM S H. Real-time object detection for autonomous driving using deep learning on edge devices. Sensors, 2021, 21(7): 2520.

[7] NAGEL M, FOURNARAKIS M, AMJAD R A, et al. A white paper on neural network quantization [Z/OL]. arXiv:2106.08295, 2021. https://arxiv.org/abs/2106.08295.

[8] NVIDIA Corporation. NVIDIA TensorRT Developer Guide. Santa Clara, CA, USA: NVIDIA Corporation, 2023.

[9] REDMON J, DIVVALA S, GIRSHICK R, et al. You only look once: Unified, real-time object detection//IEEE Conference on Computer Vision and Pattern Recognition. Las Vegas, NV, USA: IEEE, 2016: 779-788.

[10] REDMON J, FARHADI A. YOLO9000: Better, faster, stronger//IEEE Conference on Computer Vision and Pattern Recognition. Honolulu, HI, USA: IEEE, 2017: 7263-7271.

[11] XIAO B, GUO J, HE Z. Real-time object detection algorithm of autonomous vehicles based on improved YOLOv5s. Scientific Programming, 2021, 2021: 1-9.

[12] JOCHER G, CHAURASIA A, STOKEN A, et al. ultralytics/yolov5: v7.0—YOLOv5 SOTA realtime instance segmentation [Z/OL]. Zenodo, 2022. https://doi.org/10.5281/zenodo.7347926.

[13] LI H, SHEN C. Reading car license plates using deep convolutional neural networks and LSTMs [Z/OL]. arXiv:1601.05610, 2016. https://arxiv.org/abs/1601.05610.

[14] ZHERZDEV S, GRUZDEV A. LPRNet: License plate recognition via deep neural networks [Z/OL]. arXiv:1806.10447, 2018. https://arxiv.org/abs/1806.10447.

[15] XU Z, YANG W, MENG A, et al. Towards end-to-end license plate detection and recognition: A large dataset and baseline//European Conference on Computer Vision Workshops. Munich, Germany: Springer, 2018: 261-277.

[16] KAMILARIS A, BRINK C V D, KARATSIOLIS S. Training deep learning models via synthetic data: Application in unmanned aerial vehicles//IEEE International Conference on Big Data. Los Angeles, CA, USA: IEEE, 2019: 6089-6091.

[17] GHOLAMI A, KIM S, DONG Z, et al. A survey of quantization methods for efficient neural network inference [Z/OL]. arXiv:2103.13630, 2021. https://arxiv.org/abs/2103.13630.

[18] LIANG T, GLOSSNER J, WANG L, et al. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 2021, 461: 370-403.

[19] ZHOU Y, YANG K. Exploring TensorRT to improve real-time inference for deep learning//IEEE International Conference on Artificial Intelligence and Computer Applications. Dalian, China: IEEE, 2022: 201-206.

[20] AGUILAR A H, BONILLA-ROBLES J C, ZAVALA-DÍAZ J C, et al. Real-time video image processing through GPUs and CUDA and its future implementation in real problems in a smart city//IEEE International Conference on Engineering Veracruz. Boca del Río, Mexico: IEEE, 2019: 1-6.

[21] GUO H, TIAN B, YANG Z, et al. DeepStream: Bandwidth efficient multi-camera video streaming for deep learning analytics//IEEE International Conference on Multimedia and Expo Workshops. Brisbane, Australia: IEEE, 2023: 1-6.