IIETA International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Traitement du Signal
Vol. 43, No. 3, June, 2026, pp. 1287-1294
Journal homepage: http://iieta.org/journals/ts

Research on License Plate Recognition Method Based on Deep Learning and Inference Acceleration Engine

Wei Li, Yating Meng, Shijun Jiang, Xinhe Zhao, Huina Jiang*

School of Information Engineering, Beijing Institute of Petrochemical Technology, Beijing 102617, China

Corresponding Author Email: jianghuina@bipt.edu.cn

https://doi.org/10.18280/ts.430316

Received: 28 March 2026
Revised: 31 May 2026
Accepted: 13 June 2026
Available online: 30 June 2026

Keywords:
k intelligent transportation, license plate recognition, deep learning, YOLOv8 LPRNet, TensorRT, real-time video processing

ABSTRACT

To address the high inference latency of real-time license plate recognition systems in high-traffic scenarios, a TensorRT-accelerated real-time license plate recognition system is designed and implemented. YOLOv8s is adopted as the license plate detection model and License Plate Recognition Network (LPRNet) as the character sequence recognition model. During inference deployment, trained PyTorch models are exported to the Open Neural Network Exchange (ONNX) intermediate format, and TensorRT inference engines are built for both models to cut down end-to-end inference latency. An interactive web interface is constructed via Streamlit, supporting video uploading, parameter configuration, license plate query, and detection result management. All models are trained on the Chinese City Parking Dataset (CCPD) and validated on real traffic surveillance footage. Experimental results show that with TensorRT acceleration applied to both sub-models, the average end-to-end latency decreases from 14.8 ms to 9.6 ms, the frame rate (FPS) increases from 67 fps to 103 fps, and the end-to-end inference speedup ratio reaches 1.54, fully satisfying the real-time processing requirements for 1080p traffic video streams.

1. INTRODUCTION

Intelligent transportation systems (ITS) constitute the core infrastructure for modern urban traffic administration and road safety guarantee. As an essential technology for vehicle identity authentication, traffic flow counting, traffic violation snapshot capture and other practical applications, license plate recognition (LPR) governs the overall operating efficiency and supervision capability of ITS via its recognition accuracy and real-time performance [1]. As the number of urban motor vehicles keeps rising, traffic monitoring scenarios feature heavy traffic volume, multi-angle shooting and round-the-clock operation, thereby imposing higher processing performance thresholds on LPR systems.

Conventional LPR methods predominantly adopt image processing techniques such as edge detection, color segmentation and template matching. Zhao et al. demonstrated that localization algorithms built on edge features deliver low time consumption and high computational efficiency [2], yet suffer weak adaptability to blurred plate boundaries induced by stained license plates, intricate background scenes and fluctuating lighting. Conventional algorithms achieve promising recognition results under ideal circumstances with even illumination and moderate camera shooting distances. However, their robustness deteriorates severely in complicated real-world scenarios, failing to meet the demands of practical field deployment.

In recent years, the rapid advancement of deep learning techniques has delivered end-to-end solutions for license plate recognition. In the field of object detection, the YOLO (You Only Look Once) algorithm proposed by Redmon et al. transforms the detection task into a regression problem. It directly predicts bounding boxes and object categories via a single forward pass, achieving an excellent trade-off between inference speed and detection accuracy [3].

Since then, the YOLO family has undergone continuous iterative upgrades. YOLOv2 introduces batch normalization and anchor box mechanisms to further boost detection accuracy and running speed [4]. YOLOv3 adopts multi-scale prediction together with the Darknet-53 backbone network, strengthening its detection performance for small-sized objects [5]. YOLOv4 integrates multiple modules including CSPDarknet53, the Mish activation function and CIoU loss, substantially elevating detection accuracy while maintaining real-time inference capability [6]. Characterized by lightweight architecture and convenient deployment, YOLOv5 has been widely adopted in practical engineering scenarios [7]. YOLOv6 optimizes network structures specifically for industrial application scenarios and improves detection precision of small targets [8]. By extending the ELAN module and refining label assignment strategies, YOLOv7 achieves further improvements in overall detection performance [9].

The YOLOv8 model inherits the core advantages of its predecessors in the YOLO series—high detection accuracy and low time complexity. Moreover, it implements in-depth optimizations on the network backbone and feature extraction pipelines [10]. Given YOLOv8's well-balanced performance in accuracy and speed, along with its favorable compatibility for TensorRT deployment, this paper selects YOLOv8 as the

object detection model.

In the field of character recognition, Zherzdev and Gruzdev [11] proposed a lightweight convolutional neural network-based license plate recognition method named LPRNet, which realizes segmentation-free license plate recognition with an accuracy of 95%. Wang et al. [7] lightweighted YOLOv5 by integrating GP-NAS and tuned the hyperparameters of LPRNet, achieving a recognition accuracy of 98.81% on the CCPD dataset. Wang et al. [12] put forward the Fast-LPRNet algorithm; by removing fully connected layers to cut down parameter quantities, the algorithm achieves rapid recognition on FPGA devices. Silva and Jung [13] adopted cascaded YOLO networks for vehicle and license plate detection, and dramatically boosted character recognition accuracy via synthetic data augmentation. To address the recognition challenges of tilted and multi-color license plates, Xiang et al. [14] proposed a lightweight composite model combining improved YOLO and LPRNet. The proposed model maintains high recognition accuracy while compressing the model size to less than 5 MB.

Remarkable progress has been achieved in model architecture design and recognition accuracy so far. Nevertheless, most existing studies mainly focus on static images and pay insufficient attention to inference efficiency under real-time video stream scenarios. Practical traffic monitoring systems are required to continuously process high-resolution video frames. Unoptimized deep learning models generally suffer from heavy computational overhead and high inference latency, leading to low processing frame rates when deployed directly and failing to meet practical application requirements. Against this backdrop, model inference acceleration techniques are of vital importance. NVIDIA TensorRT drastically reduces inference latency with negligible accuracy loss by graph optimization, layer fusion and precision quantization. Zhang and Li [15] incorporated TensorRT into YOLOv5 for layer fusion and memory optimization, which significantly accelerated detection speed.

To tackle the abovementioned deficiency in real-time performance, this paper designs and implements a TensorRT-accelerated license plate recognition system. Taking YOLOv8s and LPRNet as the detection and recognition models respectively, the system builds individual TensorRT inference engines for both models to effectively reduce overall processing latency. Meanwhile, an interactive Web interface is constructed based on Streamlit, consisting of modules for video recognition, license plate retrieval and system management. Users can upload videos to run license plate recognition, and perform license plate query, data clearing and other operations according to the recognition outputs.

The primary contributions of this work are twofold: first, separate TensorRT inference engines are constructed for YOLOv8s and LPRNet to realize model acceleration, we systematically compare the performance gaps under different inference backend configurations, and experimental results validate the processing capacity of the dual-model acceleration scheme on real traffic videos; second, a complete end-to-end video license plate recognition pipeline is developed, covering video decoding, object detection, character recognition, CTC greedy decoding, result logging and visualized output, and a full set of user interaction functions are provided through a Streamlit-based Web application.

2. BASE MODELS AND TRAINING CONFIGURATIONS

The license plate recognition pipeline mainly consists of three sequential stages: license plate region detection, license plate image preprocessing, and character sequence recognition. The detection stage locates license plate areas within input images and outputs the coordinates of corresponding bounding boxes; the preprocessing stage crops plate regions from original images based on detection outputs and transforms cropped patches into fixed-size images via scaling, normalization and other operations; the recognition stage takes preprocessed license plate images as inputs and outputs the corresponding character sequences.

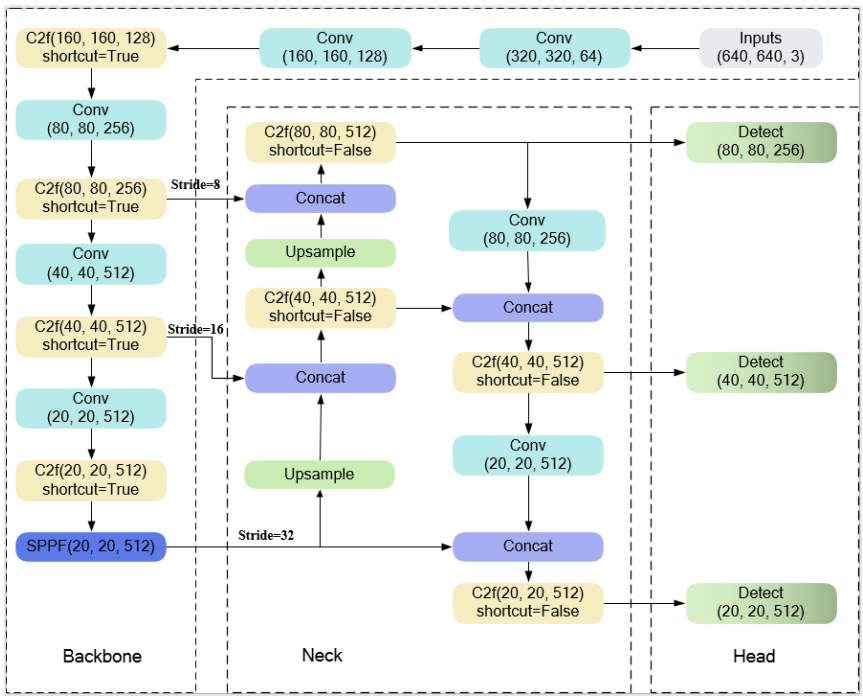

Figure 1. Schematic diagram of YOLOv8 architecture

2.1 License plate detection model

The objective of license plate detection is to accurately localize license plate regions within input images. This paper adopts YOLOv8s, the lightweight variant of YOLOv8, as the baseline detection network. Proposed by the Ultralytics team, YOLOv8 is an enhanced detection framework refined on the basis of YOLOv5, whose architecture comprises three core components: Backbone, Neck and Detection Head, as illustrated in Figure 1. The Backbone leverages the CSPDarknet architecture integrated with the C2f module to facilitate efficient feature reuse; the Neck adopts the PAN-FPN structure to achieve multi-scale feature fusion; the Detection Head decouples classification and regression tasks, which accelerates model convergence and elevates detection accuracy.

YOLOv8s achieves relatively low parameter counts while retaining competitive detection accuracy, making it suitable for tasks such as license plate detection, where target shapes are relatively fixed and stringent real-time performance is required. Compared with deeper YOLOv8 variants including YOLOv8m and YOLOv8l, YOLOv8s delivers a favorable trade-off between detection accuracy and inference speed.

In terms of training configurations, all input images are resized uniformly to a resolution of 640 X 640 pixels to accommodate the generally small proportion of license plates within full frames. The AdamW optimizer is adopted with an initial learning rate of 0.001, and a cosine annealing strategy is applied for learning rate decay. The batch size is set to 16 according to the video memory capacity of the experimental GPU. The total training epoch number is 100, and an early stopping mechanism is activated: training terminates prematurely if the validation loss fails to decline for 15 consecutive epochs, which prevents overfitting and reduces computational resource consumption. For data augmentation, Mosaic augmentation, HSV color space jitter and random horizontal flipping are utilized to boost the model robustness against varying illumination conditions and shooting perspectives. Detailed training hyperparameters are summarized in Table 1.

Table 1. Hyperparameter configuration for YOLOv8s training

Parameters	Heading 2
Input Size	640*640
Optimizer	AdamW
Initial Learning Rate	0.001
Batch Size	16
Training Epochs	100
Early Stopping Patience	15 epochs
Data Augmentation	Mosaic,HSV-Hue,Flip

2.2 License plate recognition model

The target of license plate character recognition is to decode character sequences from cropped detected license plate regions so as to obtain complete plate numbers. This work adopts LPRNet as the recognition model. This network integrates spatial feature extraction via convolutional neural networks (CNNs), sequence modeling, and end-to-end alignment based on the Connectionist Temporal Classification (CTC) loss, and its network architecture is illustrated in Figure 2.

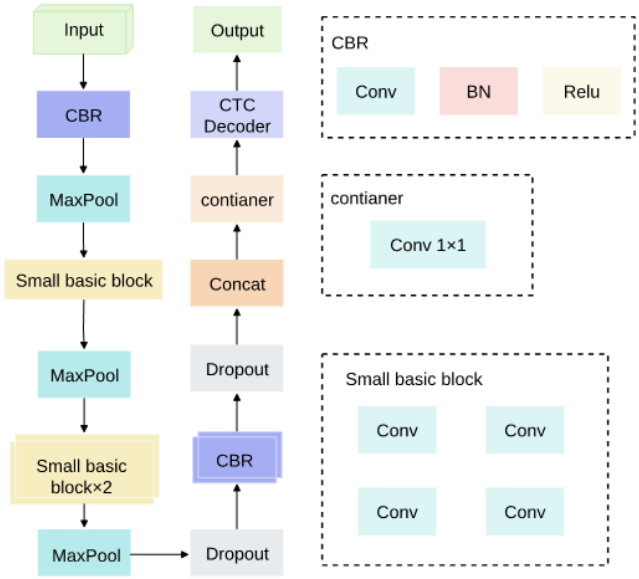

Figure 2. Architecture of License Plate Recognition Network (LPRNet)

The fundamental building block of the network adopts the CBR module, which is sequentially stacked by Convolution layers, Batch Normalization layers and ReLU activation functions to strengthen feature representation and accelerate training convergence. In the downsampling stage, a 3D max-pooling layer with a kernel size of 1 × 3 × 3 and a stride of 1 is first utilized to efficiently compress spatial dimensions while keeping the number of feature channels unchanged; afterwards, a 2D average pooling layer further aggregates spatial information to refine features and reduce computational cost. Deep feature extraction relies on the Small Basic Block, a cascade structure composed of four convolutional layers and three ReLU activation functions. Its multi-branch design strengthens the model's capability to capture subtle textures and character strokes within license plate images. To suppress overfitting, Dropout layers are inserted between LSTM layers with a retention probability of 0.8 (corresponding to a drop rate of 0.2). On this basis, the Concat operation concatenates multiple feature maps along the channel dimension to fuse multi-scale features from different layers, enriching the model's capacity to represent both global and local information of license plates. Finally, a classification container module constructed by 1 × 1 convolutions integrates and compresses extracted features, cutting down parameter quantities while enabling flexible category mapping for model outputs, followed by CTC decoding. A detailed explanation of the CTC decoding component integrated in the LPRNet model is provided as follows:

The CTC decoding stage is responsible for mapping the outputs of sequence modeling into the final character sequence. A fully connected classifier is attached after the sequence modeling module, which generates the probability distribution over all categories for each time step in the sequence. Let the sequence length be $T = 23$ and the total number of categories be $K = 66$ (including the blank token). The network output is denoted as $y = (y^1, y^2, \ldots, y^T)$, where $y^t \in R^K$ represents the raw prediction vector of the t-th frame. The category probability at each time step is defined as follows:

$$pt(k) = \frac{exp(y_t^k)}{\sum_{j=1}^{K} exp(y_t^j)}, k = 1,2,\ldots,K \quad (1)$$

CTC loss is adopted during the training phase. Given an input sequence X and ground-truth label sequence l, CTC enables flexible alignment between output sequences and labels by permitting repeated labels and blank tokens. Define a mapping function B that converts an alignment path π of length T into the final label sequence (by merging consecutive repeated characters and removing blank tokens). The conditional probability of label l equals the sum of probabilities of all paths that can be mapped to l:

$$P(l|X) = \sum_{\pi \in B^{-1}(l)} \prod_{t=1}^{T} p_t(\pi_t) \tag{2}$$

The CTC loss is defined as the negative logarithm of this conditional probability:

$$L_{CTC} = -lnP(l|X) \tag{3}$$

By optimizing this loss function, the network can learn temporal correspondences between characters without mandatory pre-alignment. During inference, the greedy decoding strategy is adopted: the category with the maximum probability is first selected at each individual time step:

$$c_t = argmax_k \, p_t \, (k), t = 1,2,\ldots,T \tag{4}$$

The label sequence $c = (c_1, c_2, \ldots, c_T)$ is obtained. A two-step post-processing procedure is then performed: adjacent identical labels are first merged to yield c' = merge_repeated (c), and all blank symbols are subsequently removed. The final output is the license plate character sequence.

$$\hat{l} = [c_t^{'} | c_t^{'} \neq blank] \tag{5}$$

The greedy decoding scheme incurs minimal computational cost, making it well suited for real-time processing. The character set is defined according to the Chinese license plate coding standard, comprising 31 abbreviations of provincial-level administrative regions, 24 English letters (with I and O excluded), and 10 Arabic numerals, totaling 65 valid characters. Together with the CTC blank symbol, the total number of output classes from the classification head is K = 66.

The training configuration for the recognition model is as follows: the Adam optimizer is employed with an initial learning rate of 5×10^{-4}, which is multiplied by 0.95 every 10 epochs. The dropout rate is set to 0.2, and the batch size is 32.

2.3 Preprocessing of license plate images

The license plate detection model outputs the bounding box coordinates of license plates, while the recognition model requires input images with fixed resolutions and normalized pixel values. Therefore, between the detection and recognition stages, the cropped plate regions extracted from the original images need to be converted into the format compatible with the recognition network. The preprocessing pipeline consists of the following three steps:

The first step is license plate region cropping. Based on the bounding box coordinates (x_1, y_1, x_2, y_2) detected by YOLOv8s, the plate region image is cropped from the original input frame. Since minor positioning deviations of bounding boxes may cut off parts of the plate edges, the box is expanded by 5 pixels in all four directions prior to cropping; meanwhile, constraints are applied to guarantee that the expanded coordinates do not exceed the boundaries of the original image:

$$x_1^{'} = max(0, x_1 - 5) \tag{6}$$

$$y_1^{'} = max(0, y_1 - 5) \tag{7}$$

$$x_2^{'} = max(W, x_2 + 5) \tag{8}$$

$$y_2^{'} = max(H, y_2 + 5) \tag{9}$$

where, W and H denote the width and height of the original image, respectively. The expanded region (x_1', y_1', x_2', y_2') corresponds to the final area to be cropped.

The second step is size resizing: all cropped license plate images are uniformly resized to a resolution of 94 × 24 pixels. This dimension matches the input layer specification of LPRNet, and its aspect ratio approximates the physical proportion of standard Chinese license plates, which preserves fine character details while preventing excessive geometric distortion. Bilinear interpolation implemented in OpenCV is adopted for the resizing operation.

The final step is normalization: the data type of resized images is converted from uint8 to float32, followed by division by 255.0 to map all pixel values into the range [0, 1]:

$$I_{nom} = I_{resized}/255.0 \tag{10}$$

Afterwards, the channel order of the image is transformed from HWC (Height-Width-Channel) to CHW (Channel-Height-Width) to match the input format required by the PyTorch and TensorRT inference engines. The processed image obtained above is then fed into the network for character recognition.

3. TENSORRT INFERENCE ENGINE

TensorRT is a deep learning inference accelerator developed by NVIDIA, which is designed to boost the inference performance of deep learning models. It mainly serves to accelerate model execution speed, reduce inference latency and increase throughput. Supporting multiple deep learning frameworks such as PyTorch and TensorFlow, TensorRT provides programming interfaces for both C++ and Python to achieve highly efficient inference workflows.

TensorRT inference mainly consists of two phases: engine construction and inference execution using the built engine. There are two approaches for the inference engine construction phase. The first approach is to manually call TensorRT APIs to reconstruct the network layer by layer. This method delivers great flexibility and high precision, yet it involves heavy workload and inconvenient subsequent modification. The second approach first exports the trained model to Open Neural Network Exchange (ONNX) format, then converts the ONNX model into an inference engine via TensorRT and serializes it for storage. This scheme is more convenient and efficient, and TensorRT provides comprehensive compatibility support for the ONNX format. This paper adopts the second method to build the inference engine, and the general workflow of engine construction is illustrated in Figure 3.

Figure 3. Architecture of License Plate Recognition Network (LPRNet)

3.1 Open Neural Network Exchange model export and compatibility processing

Exporting the PyTorch model to ONNX format is a prerequisite for building the TensorRT engine. As a mature detection framework, YOLOv8s has all operators within its network architecture well-supported by ONNX, leading to a smooth export process. In contrast, LPRNet incorporates bidirectional LSTM layers; early versions of ONNX opsets offered limited support for such dynamic recurrent structures. Multiple debugging attempts were conducted before we obtained a fully intact ONNX model that can be correctly parsed by TensorRT. Furthermore, to enable variable batch processing during subsequent inference, the batch dimension of the input tensor is defined as a dynamic axis upon export. This configuration allows a single compiled engine to handle inference requests with varying batch sizes.

3.2 TensorRT engine construction

After converting the model into ONNX format, we proceed to construct the TensorRT inference engine. The engine construction procedures for both the detection model and the recognition model follow the steps listed below, and the overall engine building pipeline is depicted in Figure 4.

(1) Builder Initialization: Instantiate a TensorRT builder that acts as the control hub for the entire optimization pipeline. Meanwhile, initialize a logger to track status messages and exceptions generated during the construction process.

(2) Network Parsing: Load the ONNX model file via the ONNX parser and convert it into TensorRT's internal network representation. This step fully restores operator types, inter-layer connections and weight parameters within the computational graph, and verifies that all operators are supported by TensorRT.

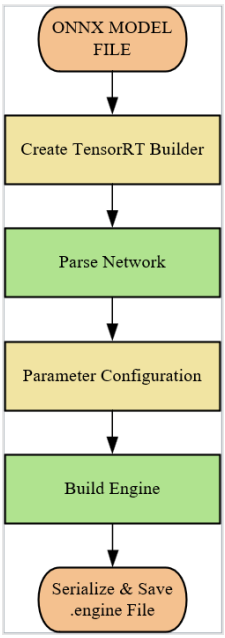

Figure 4. Flowchart of engine construction

(3) Parameter Configuration: Configure optimization parameters according to the hardware specifications of the target GPU. Key configurations include: setting the maximum workspace size to balance memory footprint and optimization strength; selecting a precision mode to maximize inference speed while keeping accuracy loss within an acceptable range; specifying ranges for dynamic dimensions so the generated engine can handle inference requests with various batch sizes.

(4) Engine Building: TensorRT executes a series of automatic graph optimizations, such as layer fusion — merging successive convolution, batch normalization and activation layers into a single kernel function. Once optimization finishes, a deployable inference engine is generated.

(5) Serialization and Storage: Serialize the engine object and save it as a .engine file. This file stores the optimized computational graph structure, fused operator implementations and all weight parameters. During runtime, the engine can be directly loaded via deserialization without repeated parsing and optimization.

4. SYSTEM DESIGN

4.1 Overall architecture

This system adopts a modular layered architecture with separated front-end and back-end, which is divided into three tiers: the front-end interaction layer, the main scheduling layer and the inference execution layer. The overall architecture is illustrated in Figure 5. The three layers perform their respective functions and operate collaboratively. By decoupling user interaction, task scheduling and GPU inference, the system balances convenient visual operation and high-efficiency batch video processing.

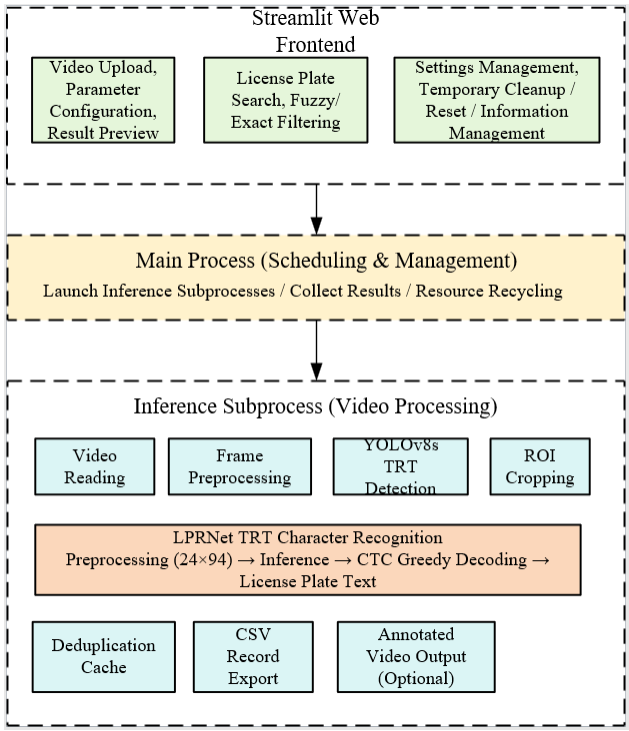

Figure 5. Overall system architecture

The top tier is the Streamlit Web front-end interaction layer, which builds a lightweight graphical user interface accessible

via web browsers without client installation. This layer integrates multiple management functions including video upload, inference parameter configuration, recognition result preview and statistical display, multi-condition license plate retrieval (supporting fuzzy matching, exact matching and regular expression matching, with secondary filtering by confidence intervals and timestamp ranges), as well as system configuration and cache clearance. It is responsible for issuing recognition tasks and rendering parsed results.

The middle tier is the main scheduling process running on the main Python thread. It receives operation commands from the front-end and launches independent child inference processes via the subprocess module to realize asynchronous decoupling between front and back ends. As a result, the web interface remains responsive during heavy computation and will not be blocked by long-running inference tasks. After spawning a child process, the main process continuously monitors its running status. Upon task completion, it automatically recycles occupied resources, loads CSV recognition logs and optional annotated videos into memory, formats the data, and pushes formatted content to the front-end for display. Additionally, the main process implements exception capturing and timeout control to guarantee overall system robustness.

The bottom tier is the inference execution layer, which encapsulates a complete video processing pipeline where all compute-intensive tasks are carried out. Centered on two TensorRT inference engines of YOLOv8s and LPRNet, all inference computations are accelerated on the NVIDIA RTX 4060 GPU. This layer executes the full workflow: frame-by-frame video decoding, image preprocessing, license plate detection, ROI cropping, character recognition, CTC decoding, duplicate removal cache, and persistent storage of inference results.

As shown in Figure 5, video frames are sequentially processed within the inference child process through reading, preprocessing, YOLOv8s object detection, ROI cropping, LPRNet character recognition and CTC decoding. After duplicate filtering and caching, the final license plate recognition results are output. Driven by TensorRT inference engines, both detection and recognition tasks run sequentially on the same GPU. The pipelined frame processing mechanism effectively reduces the latency of single-frame processing. This architecture decouples front-end interaction, task scheduling and computational execution with clear functional boundaries and standardized data interfaces for each module. It not only improves system maintainability, but also reserves extensibility for introducing more complex recognition logic or multi-channel concurrent video processing in future iterations.

4.2 Video processing

Complete Processing Pipeline for a Single Video:

(1) Video Frame Reading: Read frames from the input video frame by frame via OpenCV, and acquire basic metadata such as frame rate and resolution.

(2) Frame Preprocessing: The image is scaled up proportionally to boost detection performance for small-sized license plates; meanwhile, the maximum resolution is capped to avoid GPU memory overflow.

(3) License Plate Detection: Feed the preprocessed frames into the TensorRT engine of YOLOv8s to obtain bounding boxes and corresponding confidence scores. The coordinates of detection boxes are mapped back to the original resolution according to the scaling ratio, and each bounding box is expanded by 5 pixels on all four sides to fully cover the license plate area.

(4) Character Recognition: Crop out the license plate region and feed it into the TensorRT engine of LPRNet for preprocessing, inference and greedy decoding to generate the license plate string.

(5) Deduplication and Logging: An ordered dictionary is used to record detected license plates along with their latest timestamp. If the same plate reappears within the deduplication interval, subsequent detections are discarded to prevent redundant records for the same vehicle. Validated records containing frame index, timestamp, plate text and confidence value are written into a CSV file.

(6) Result Visualization (Optional): If annotated output video is required, draw bounding boxes and license plate text on the original frames using TrueType fonts from PIL. Real-time FPS values are overlaid at the top-left corner, and the annotated stream is finally encoded and exported as a video file.

After the completion of video processing, all relevant resources are released and garbage collection is triggered to eliminate memory access exceptions that may occur during TensorRT engine destruction. The main process guarantees full isolation and safe recycling of GPU resources between different task invocations.

5. EXPERIMENT VERIFICATION

All experiments are conducted under the Python environment, and the detailed hardware and software configurations are listed in Table 2.

Table 2. Software and hardware configuration

Item	Specification
CPU	Intel(R) Core(TM) i7-14650HX
System Memory	16GB
GPU	NVIDIA GeForce RTX 4060
GPU VRAM	8GB
Operating System	Ubuntu 24.04
Python	3.10
Deep Learning Framework	Pytorch 2.1.2
CUDA	11.8
TensorRT	8.6.1

5.1 Dataset

The CCPD dataset is adopted for model training. This dataset contains more than 250,000 images of Chinese urban license plates, covering various complex scenarios such as blurriness, tilt angles, rainy weather and snowy weather. The CCPD dataset does not provide separate annotation files; instead, annotation information is embedded in the filename of each image. Each segment of the filename corresponds to regional code, horizontal and vertical tilt angles, bounding box coordinates, corner point coordinates, license plate character encoding, brightness value, blurriness and other attributes. Annotation files can be generated by parsing these filenames. To guarantee the validity and scientific rigor of experiments, 10,000 images are randomly selected as the training set, and another 2,000 images form the test and validation set.

A self-collected surveillance video captured at urban traffic

intersections is used for system performance evaluation. The video lasts approximately 901 seconds with a total of 27,030 frames, a resolution of 1920 × 1080 and a frame rate of 30 FPS. The scene contains passing vehicles captured at varying distances and shooting angles.

5.2 Evaluation metrics

This experiment mainly evaluates the processing speed of the system under high-traffic scenarios, and the assessment is carried out from the following three dimensions:

•FPS (Frames Per Second): The number of frames that the system can process and output per second. A higher FPS value indicates faster processing speed.

•Average Latency: The average time consumed by the system from receiving an input frame to finishing processing and outputting results, measured in milliseconds (ms). A lower average latency corresponds to faster response performance.

•Speedup Ratio: The ratio between the actual duration of the video and the time taken by the system to process the entire video, which reflects how many times faster the processing speed is compared to real-time playback.

A self-collected surveillance video captured at urban traffic intersections is used for system performance evaluation. The video lasts approximately 901 seconds with a total of 27,030 frames, a resolution of 1920 × 1080 and a frame rate of 30 FPS. The scene contains passing vehicles captured at varying distances and shooting angles.

5.3 Evaluation of TensorRT acceleration performance

To verify the acceleration performance of TensorRT, four groups of comparative experiments under identical test constraints are designed: Group A adopts the original PyTorch framework for both detection and recognition models; Group B only replaces the detection model backend with TensorRT while retaining PyTorch for the recognition branch; Group C only accelerates the recognition model via TensorRT and keeps the detection model running on PyTorch; Group D implements TensorRT acceleration for both detection and recognition models.

All test groups apply unified FP16 quantization and run under a single-process pipeline. The recorded end-to-end latency covers image preprocessing, model inference, and result post-processing.The speedup ratio is calculated as the end-to-end latency of baseline Group A divided by the end-to-end latency of each experimental group.The experimental results are presented in Table 3.

Table 3. Single-process performance comparison of different inference combinations

Item	Det. Model	Rec. Model	Latency /ms	FPS	Speedup Ratio
A	PyTorch	PyTorch	14.8	67	1.0
B	TensorRT	PyTorch	11.8	84	1.25
C	PyTorch	TensorRT	13.4	74	1.10
D	TensorRT	TensorRT	9.6	103	1.54

As can be seen from Table 3, comparing Group A and Group B (only detection model accelerated by TensorRT), the average latency is reduced by 20.3% and FPS increases by 25.4%. When only the recognition model is converted to TensorRT acceleration (Group A vs. Group C), the average latency drops by 9.5%, accompanied by a 10.4% FPS improvement. After both detection and recognition models are optimized with TensorRT (Group A vs. Group D), the average end-to-end latency is further reduced to 9.6 ms, the frame rate reaches 103, and the overall speedup ratio rises to 1.54. This proves that the graph optimization and FP16 quantization modules of TensorRT deliver obvious acceleration gains for both sub-models. The fully accelerated system achieves a frame rate far higher than the widely recognized industrial real-time threshold of 30 FPS for 1080p traffic surveillance videos, realizing ultra-real-time processing capacity.

With only the inference backend replaced and zero modifications to the original architectures of YOLOv8s and LPRNet, the overall end-to-end latency is reduced by approximately 35%, and FPS is improved by around 54%. The speedup ratio increases from 1.00 (baseline Group A) to 1.54, which indicates that the frame processing time is shortened by roughly one-third, greatly improving the overall throughput of the video license plate recognition system.

5.4 Visualization verification

The proposed system is adopted to perform license plate recognition on high-traffic surveillance video captured by monitoring cameras. The corresponding visualization results are illustrated in Figure 6, which demonstrates the processing performance of the system under real-world traffic scenarios.

Figure 6. License plate recognition in traffic scenes

6. CONCLUSIONS

Aiming at excessive end-to-end inference latency of license plate recognition pipelines for real-time traffic monitoring, this paper designs and implements a TensorRT-accelerated real-time license plate recognition system. The proposed pipeline employs YOLOv8s and LPRNet as the license plate detection and character recognition sub-models, respectively. By converting both trained PyTorch models into optimized TensorRT inference engines without modifying the original network architectures, the overall end-to-end processing latency is significantly reduced. A user-friendly interactive web dashboard is further developed based on Streamlit, which integrates practical functions including video upload, license plate query, and detection result management for offline verification and demonstration.

Comparative experiments verify the acceleration effectiveness of the dual-model TensorRT optimization strategy: the average end-to-end frame latency drops from 14.8 ms to 9.6 ms, representing a latency reduction of 35.1%. Correspondingly, the frame rate increases from 67 FPS to 103 FPS, with the standardized end-to-end speedup ratio reaching 1.54. The optimized system far exceeds the industrial minimum real-time threshold of 30 FPS for 1080p traffic

surveillance streams and achieves ultra-real-time processing capability. Moreover, the TensorRT backend replacement introduces no degradation to license plate detection and recognition accuracy on both the CCPD dataset and real-world road monitoring footage, enabling low-latency, end-to-end real-time license plate recognition while retaining full recognition performance.

The current system is only validated on desktop GPUs under single-process testing conditions, and long-term stability on embedded edge devices has not been fully verified. For follow-up research, we will migrate the entire pipeline to edge computing hardware such as NVIDIA Jetson Orin to evaluate its embedded acceleration performance. We will also conduct long-duration stability tests under complex actual road traffic scenarios, and further optimize the pipeline for large-scale distributed deployment on multiple edge monitoring nodes.

REFERENCES

[1] Song, J.H., Xia, B., Zhao, Y.W., Liu, X.Y. (2025) License plate recognition algorithm based on improved YOLOv8 and LPRNet. Journal of Shenyang Ligong University, 2025, 44(3): 39-46. https://doi.org/10.3969/j.issn.1003-1251.2025.03.006

[2] Zhao, W., Zhang, N.N. (2019). Application of deep learning in license plate locating algorithm in complicated environment. Modern Electronics Technique, 42(17): 38-42. https://doi.org/10.16652/j.issn.1004-373x.2019.17.009

[3] Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 779-788. https://doi.org/10.1109/CVPR.2016.91

[4] Redmon, J., Farhadi, A. (2017). YOLO9000: Better, faster, stronger. In 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 6517-6525. https://doi.org/10.1109/CVPR.2017.690

[5] ul Haq, Q.M., Ruan, S.J., Haq, M.A., et al. (2021). An incremental learning of YOLOv3 without catastrophic forgetting for smart city applications. IEEE Consumer Electronics Magazine, 11(5): 56-63. https://doi.org/10.1109/MCE.2021.3096376

[6] Humayun, M., Ashfaq, F., Jhanjhi, N.Z., Alsadun, M.K. (2022). Traffic management: Multi-scale vehicle detection in varying weather conditions using yolov4 and spatial pyramid pooling network. Electronics, 11(17): 2748. https://doi.org/10.3390/electronics11172748

[7] Wang, F., Tang, Z.R., Zou, J.Y., Wang, H.B., Xing, G.X. (2023). Improved end-to-end license plate recognition based on GP-NAS. Journal of Chongqing University of Technology (Natural Science), 37(10): 38-46. https://doi.org/10.3969/j.issn.1674-8425(z).2023.10.005

[8] Norkobil Saydirasulovich, S., Abdusalomov, A., Jamil, M.K., Nasimov, R., Kozhamzharova, D., Cho, Y.I. (2023). A YOLOv6-based improved fire detection approach for smart city environments. Sensors, 23(6): 3161. https://doi.org/10.3390/s23063161

[9] Qiu, Y., Lu, Y., Wang, Y., Jiang, H. (2023). IDOD-YOLOv7: Image-dehazing YOLOv7 for object detection in low-light foggy traffic environments. Sensors, 23(3): 1347. https://doi.org/10.3390/s23031347

[10] Lu, L., Li, M.L., Xiong, W., Gong, K., Ma, H., Zhang, X. (2025). Infrared image detection of substation equipment based on improved YOLOv8. Infrared Technology, 47(9): 1135-1141.

[11] Zherzdev, S., Gruzdev, A. (2018). LPRNET: License plate recognition via deep neural networks. arXiv preprint arXiv:1806.10447. https://doi.org/10.48550/arXiv.1806.10447

[12] Wang, Z., Jiang, Y., Liu, J., Gong, S., Yao, J., Jiang, F. (2021). Research and implementation of fast-LPRNet algorithm for license plate recognition. Journal of Electrical and Computer Engineering, 2021(1): 8592216. https://doi.org/10.1155/2021/8592216

[13] Silva, S.M., Jung, C.R. (2020). Real-time license plate detection and recognition using deep convolutional neural networks. Journal of Visual Communication and Image Representation, 71: 102773. https://doi.org/10.1016/j.jvcir.2020.102773

[14] Xiang, M.L., Yu, L.D., Meng, H.L., Du, W.J., Yang, X.Y. (2024). A lightweight model for identifying tilted mixed color license plates. Journal of Beijing University of Posts and Telecommunications, 47(5): 92-98.

[15] Zhang, X., Li, B. (2025). Tennis ball detection based on YOLOv5 with tensorrt. Scientific Reports, 15(1): 21011. https://doi.org/10.1038/s41598-025-06365-3