Software System Design

VIROC [Vehicle Identification OCR]

Status: Draft

Date: 07 05 2025

Table of Contents

Introduction

This document outlines the system design for **VIROC [Vehicle Identification OCR]**. It details the architectural decisions, components, data models, and operational considerations for the project.

Overview

The purpose of the VIROC system is to provide a platform for vehicle license plate OCR that allows users to submit images that contain license plates and receive back result license plate text and associated confidence scores. It is intended for use in an application stack that requires reading license plate data from live camera views to enhance customer experience and operational efficiency.

Goals and Objectives

- **Goal 1:** To provide a scalable platform for vehicle license plate detection and OCR.
- **Goal 2:** To ensure high accuracy in license plate recognition.
- **Goal 3:** To deliver a user-friendly interface for data submission and results retrieval.

Non-Goals

Model training is out of scope for this project. The system will utilize pre-trained models for license plate detection and OCR.

Glossary

- **API:** Application Programming Interface
- **OCR:** Optical Character Recognition

Requirements

This section details the functional and non-functional requirements that the system must satisfy.

Functional Requirements

Describe the specific behaviors and functions of the system. Use a numbered list for clarity.

1. **Submission API** - The system must provide an API endpoint for users to submit images containing vehicle license plates returning JSON formatted results.

Problem Cases

The system should handle clear images of:
- **Case 1:** a single license plate.
- **Case 2:** multiple license plates.
- **Case 3:** no license plates.

The system may not be able to handle images with:
- **Case 4:** low resolution or poor lighting conditions.
- **Case 5:** obscured or damaged license plates.
- **Case 6:** licenses not attached to vehicles, such as those on a wall or in a parking lot.

High-Level Architecture

This section provides a bird's-eye view of the system's architecture.
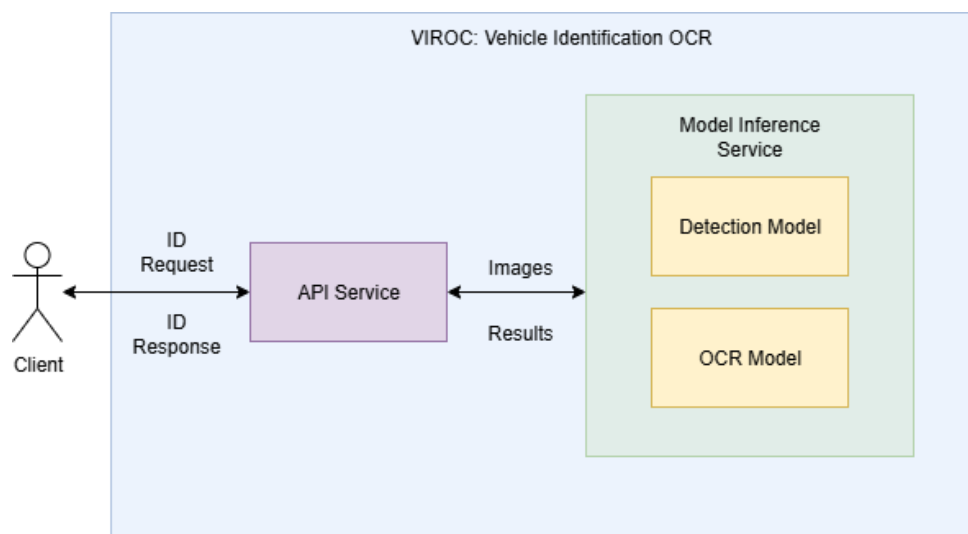
Architectural Diagram

Figure 1: High-Level System Architecture.

System Components

- **API Service:** A FastAPI service that handles incoming requests, processes images, and interacts with the model inference service.
- **Model Inference Service:** A service that uses NVIDIA TritonServer to perform inference on pre-trained models for license plate detection and OCR.
- **Detection Model**: A model that detects license plates in images.
- **OCR Model**: A model that performs OCR on detected license plates to extract text.

Technology Stack

- **Backend:** Python, FastAPI
- **Model Inference:** NVIDIA TritonServer, Models from HuggingFace
- **Infrastructure:** Docker

Limitations and Potential Improvements

Benchmarks are required to determine the performance of the system under load and also to determine production accuracy.

Successes
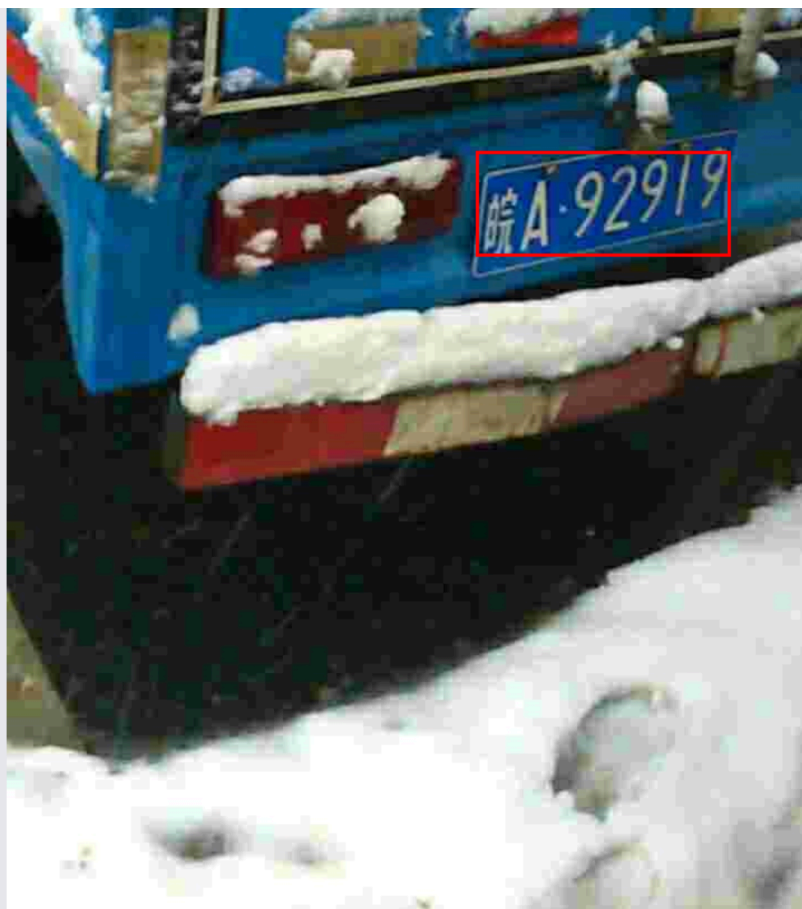
Generated text: 皖A·92919

Failures

Generated text: HEARN626

AI Usage Disclosure

Gemini 2.5 Pro generated the initial file for the system design template.

Github Copilot autocomplete was used during development of python scripts.