# Introduction

The Nepali Number Plate Recognition System is an intelligent computer vision-based application designed to automatically detect and recognize vehicle license plates used in Nepal. The goal of this system is to simplify and automate processes such as vehicle verification, traffic monitoring, and digital record-keeping by converting images of vehicles into readable plate numbers. Unlike standard systems that are built for English alphanumeric plates, our system is specifically tailored to handle Nepali license plates, which often include Devanagari characters, regional codes, and embossed text formats.

The architecture of our system is divided into two main phases: object detection (OD) and optical character recognition (OCR). For the detection phase, we employed the lightweight and efficient YOLOv8n model, which was trained on a curated dataset from Kaggle containing images of Nepali vehicles and plates. This model accurately locates the number plate region within an image. Once detected, the system crops the plate and passes it to the OCR component. For character recognition, we use EasyOCR, a deep learning-based text recognition library that supports Devanagari script, enabling our system to read and extract the actual characters from the license plate image.

# Objectives

The main objectives of Nepali License Plate Recognition System include:

* To detect number plates from input images, videos, or live camera feed using YOLOv8.
* To recognize text (both Nepali and English) on the detected number plate using OCR.
* To implement a web-based user interface using Flask for easy access to the system.

# Analysis

## Strategic Assessment

The Nepali Number Plate Recognition System (NNPRS) has been developed with a vision to modernize vehicle tracking and regulation in Nepal, where traffic monitoring is still largely dependent on manual efforts. This project aligns closely with several national and organizational objectives:

* Alignment with Governmental and Organizational Goals:
* The Government of Nepal is increasingly adopting digitization and automation in public services, including the transportation sector. This project provides a foundation for automating vehicle monitoring and enforcement of traffic regulations.
* Government agencies such as the Department of Transport Management (DoTM), Metropolitan Traffic Police Division, and highway authorities can integrate this system to improve surveillance and record-keeping accuracy.
* Strategic Impact:
* NNPRS helps reduce human errors and workload associated with manual checking of license plates.
* It supports digital evidence generation for law enforcement, such as tracking stolen vehicles or identifying traffic rule violations.
* By contributing to road safety and efficient toll collection, the system supports national goals related to sustainable urban mobility and smart infrastructure development.
* Market Demand and Potential:
* There is a clear and growing demand in the Nepali market for intelligent transportation systems (ITS). Current solutions available in Nepal are either fully manual or use foreign software that lacks localization support (e.g., Devanagari OCR).
* With increased urbanization and vehicle growth in Nepal, the need for scalable automated systems is evident.
* The uniqueness of this system lies in its ability to read both English and Nepali license plates, a feature rarely supported by existing systems. This gives it a competitive edge for wide adoption in smart city initiatives.
* Scalability:
* The system is modular and can be expanded to include real-time monitoring, integration with cloud databases, and even alert systems for law enforcement.

In conclusion, this project not only supports the strategic goals of modernization and digitization in the Nepali transportation sector but also addresses a significant gap in the current infrastructure by introducing script-specific OCR functionality.

## Technical Assessment

The NNPRS is technically feasible and incorporates various software components integrated into a modular system. Here’s how the technical components are structured:

**Frontend**

* Developed using Flask and Jinja2 for a minimal web interface.
* Allows users to upload images or videos and view the extracted plate numbers.
* The output is displayed on the result page after detection and recognition.

**Backend**

* Written entirely in Python.
* Handles:
  + Uploading image or video files.
  + Processing them using YOLOv8 for license plate detection.
  + Running EasyOCR for character recognition.
  + Storing results and handling file organization.
* The backend supports multiple routes: /image, /video, and /realtime to handle different input types.

**Storage (CSV & Image Storage)**

* The extracted results (vehicle number, time, file name, confidence, etc.) are saved in a CSV file.
* The detected plate region is also saved as a raw image file (typically .jpg or .png format).
* This allows for future auditing, training dataset expansion, and result verification.

**Machine Learning**

* YOLOv8 is used for detecting license plates from various input formats.
* EasyOCR is used for recognizing the characters in both English and Devanagari.
* The dataset includes Nepali license plates manually collected and labeled.
* The system is trained and tested using real-world Nepali traffic images.

**Deployment**

* The system is currently deployed locally on a PC.
* Raw image and CSV results are generated instantly on detection.
* For long-term deployment, it can be containerized using Docker or moved to cloud platforms for broader accessibility.

**Technical Challenges**

* Handling blurred, angled, or occluded license plates.
* Ensuring accurate recognition of Devanagari script, as pre-trained OCR models have limited accuracy for Nepali.
* Large image and video file sizes affecting real-time performance.
* Ensuring accurate file handling and non-duplication while storing results.
* Developing a consistent preprocessing pipeline to ensure image quality is sufficient for OCR.

## Economic Analysis

### Present Worth, Future Worth and Annual Worth

|  |  |  |  |
| --- | --- | --- | --- |
| Year | Investment (NPR) | Benefit (NPR) | Net Cash Flow |
| 0 | -200,000 | 0 | -200,000 |
| 1 | 0 | 100,000 | 100,000 |
| 2 | 0 | 120,000 | 120,000 |
| 3 | 0 | 130,000 | 130,000 |
| 4 | 0 | 40,000 | 40,000 |
| 5 | 0 | 50,000 | 50,000 |

Interest rate (i) = 10% = 0.10

1. Present Worth

Formula:

PW =

PW =

=

=

= 146072

Since the Present value is positive, it means that investment is expected to be a good investment.

1. Future Worth

Formula:

FW =

Where:

FW = Future Worth

PW = Present Worth (in our case: 146,072 NPR)

r = interest rate (10% = 0.10)

n = number of years (5)

FW =

=

=

Since our final worth is positive. It indicates that our initial investment of NPR 200,000 is not only recovered but also results in a profitable return over the project’s lifetime.

1. Annual Worth

Formula:

PW =

PW =

=

=

= 146072

### Internal Rate of Return

Formula:

IRR=

# Activity planning and Scheduling

## Work Breakdown Structure (WBS)

The Work Breakdown Structure (WBS) for the project divides the full project scope into manageable tasks. Here’s a hierarchical WBS for our system:

Level 1: Nepali Number Plate Recognition System

1.1 Requirement Analysis

1.1.1 Functional and Non-Functional Requirements

1.1.2 Dataset Needs Identification

1.2 Dataset Collection and Preparation

1.2.1 Collect Images of Nepali Plates

1.2.2 Label and Annotate Dataset

1.2.3 Split Dataset (Train/Test)

1.3 Model Development

1.3.1 Train YOLOv8 for Plate Detection

1.3.2 Integrate EasyOCR for Character Recognition

1.3.3 Testing and Tuning

1.4 System Integration

1.4.1 Build Flask Web App

1.4.2 Connect Detection and OCR Pipeline

1.4.3 Display Results and Save Output

1.5 Deployment and Testing

1.5.1 Test with Real Images and Videos

1.5.2 Store Results in CSV and Images

1.5.3 Debugging and Performance Optimization

1.6 Documentation and Presentation

1.6.1 Report Writing

1.6.2 Mid and Final Defense Preparation

## Bar Chart

## Network Planning Model

The network planning model identifies task sequences and dependencies. It helps manage timelines and avoid delays.

Key Dependencies:

* Dataset annotation must be completed before training.
* Model development must be finished before integration.
* Integration must be complete before system testing.

## Critical Path method

The Critical Path Method (CPM) helps identify the longest path of dependent activities — meaning if any of these are delayed, the whole project is delayed.

Defined Activities with Estimated Durations

|  |  |  |  |
| --- | --- | --- | --- |
| Activity | Description | Duration (Days) | Dependency |
| A | Requirement Analysis | 2 | - |
| B | Data Collection & Annotation | 5 | A |
| C | Model Training (YOLOv8 + OCR) | 7 | B |
| D | Flask Integration & UI | 4 | C |
| E | Testing & Deployment | 3 | D |
| F | Final Report & Presentation | 3 | E |

CPM Path:

Critical Path = A → B → C → D → E → F

Total Duration = 2 + 5 + 7 + 4 + 3 + 3 = 24 days

All tasks in this path are critical activities — delay in any of them will delay the entire project.

## Program Evaluation and Review Technique (PERT)

# Risk Management

Risk management is a critical component of software project planning. It ensures you can identify, assess, and prepare for potential challenges that could affect project success.

## Nature and Identification of Risk

In this project, we identified various technical, operational, and resource-related risks. The nature of these risks is explained below:

|  |  |
| --- | --- |
| Risk Type | Description |
| Technical Risk | Difficulty in achieving high OCR accuracy for Devanagari script; dependency on model performance. |
| Data Risk | Limited availability of high-quality annotated images of Nepali number plates. |
| Integration Risk | Bugs during integration of YOLOv8 detection with EasyOCR and Flask interface. |
| Hardware/Performance Risk | Real-time processing may slow down due to hardware limitations (especially without GPU). |
| Human Resource Risk | If one developer is unavailable or delayed, it can impact model training or integration timelines. |
| Deployment Risk | Challenges while deploying the system on the web/cloud due to dependencies or compatibility issues. |
| Security Risk | If not handled carefully, storing results (CSV/images) could expose sensitive vehicle data. |

## Risk Evaluation & Evaluation of risk to the schedule using Z-values

# Resource Allocation

Resource allocation ensures that all necessary human, technical, and financial resources are properly identified, assigned, and balanced throughout the project lifecycle.

## Identifying Resource Requirements

Here are the key resources required for our project:

**Human Resources**

2 Developers:

* Sadik Karki (Backend – YOLOv8, EasyOCR integration)
* Kushala Shrestha (Frontend – Flask UI, CSV/Image output handling)

1 Supervisor/Reviewer: For guidance, feedback, and approval

1 Data Annotator (shared): Helps label license plate datasets for model training

**Technical Resources**

* Personal computers with at least 8GB RAM and 2GB VRAM
* GPU access (local or cloud, e.g., Google Colab)
* Camera or video input for real-time testing
* Image datasets (Nepali license plates)
* Python environment (with libraries: YOLOv8, OpenCV, EasyOCR, Flask)

**Financial Resources**

|  |  |
| --- | --- |
| Resource | Estimated Cost (NPR) |
| Internet & power usage (training) | 5,000 |
| Data collection (travel/equipment) | 15,000 |
| GPU/Cloud rental (if needed) | 20,000 |
| Miscellaneous (documentation, UI) | 5,000 |
| Total | 45,000 |

## Resource Allocation

**Human Resource Allocation**

|  |  |  |
| --- | --- | --- |
| Member | Role | Responsibility |
| Sadik Karki | Backend & Model Developer | YOLO v8 training, OCR integration, real-time |
| Kushala Shrestha | Frontend & Integration Lead | Flask UI, result storage, output formatting |

**Software & Tool Allocation**

|  |  |
| --- | --- |
| Tool | Purpose |
| YOLOv8 + Ultralytics | License plate detection model |
| EasyOCR | Optical Character Recognition |
| OpenCV | Image preprocessing & video handling |
| Flask + Jinja2 | Web interface for upload/output |
| GitHub | Code collaboration and version control |

**Financial Allocation**

|  |  |
| --- | --- |
| Phase | Estimated Budget (NPR) |
| Data preparation | 10,000 |
| Model development | 20,000 |
| Testing and deployment | 5,000 |
| Report & documentation | 5,000 |
| Buffer/emergency cost | 5,000 |

## Resouce Smoothing and Resource Balancing

To avoid overloading certain team members or tools, the following approaches were used:

**Resource Smoothing**

* Staggered tasks: Instead of doing all tasks sequentially, non-dependent tasks were arranged in parallel (e.g., model training and UI design).
* Time buffer: Extra time added between phases (like after model training) to prevent overrun.

**Resource Balancing**

* Team responsibilities were balanced to avoid dependency on a single person.
* Used Google Colab for training models instead of heavy local compute, saving time and cost.
* Training and testing split evenly over weeks to avoid performance overload.

# Monitoring and Control

Monitoring and control ensure the project is progressing as planned. It helps track schedule, cost, scope, and quality, and take corrective actions if necessary.

## Collecting Data

In our project, data collection refers to both:

1. Project Progress Data:

* Weekly log sheets documenting tasks completed (e.g., dataset preparation, training, UI updates).
* GitHub commits and update history for coding milestones.
* Supervisor meeting notes and feedback.

1. System Data (Output):

* Result CSV files storing recognized plate numbers, timestamps, and confidence.
* Raw images of detected number plates saved for validation.
* Logs of model accuracy, recognition rate, and failure cases.

## Visualizing Progress

## Cost (Expenditure) Monitoring

We monitored the actual cost spent vs. the planned budget to ensure the project stays within financial limits.

Estimated Budget vs Actual Cost (in NPR):

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Estimated Cost | Actual Cost | Status |
| Dataset Collection | 15,000 | 13,000 | Under |
| Model Training | 20,000 | 21,500 | Over |
| Testing/Integration | 5,000 | 4,500 | On Track |
| Miscellaneous | 5,000 | 4,000 | Under |
| Total | 45,000 | 43,000 | Within Budget |

## Earned Value Analysis

Earned Value Analysis (EVA) helps evaluate project performance in terms of cost and schedule using three key metrics:

* PV (Planned Value): Work scheduled/planned till now.
* EV (Earned Value): Value of work actually completed.
* AC (Actual Cost): Actual money spent.

Example (Midway - Week 5):

Metric Value (NPR)

|  |  |
| --- | --- |
| PV | 25,000 |
| EV | 20,000 |
| AC | 23,000 |

Calculations:

Schedule Performance Index (SPI) = EV / PV = 20,000 / 25,000 = 0.8

→ Project is behind schedule.

Cost Performance Index (CPI) = EV / AC = 20,000 / 23,000 ≈ 0.87

→ Project is over budget (slightly).

Summary:

|  |  |  |
| --- | --- | --- |
| Metric | Value | Interpretation |
| SPI | 0.8 | Behind schedule |
| CPI | 0.87 | Slightly over cost |
| EV < PV | Work done < Work planned |  |
| EV < AC | Cost > Value of work done |  |

# Testing

Testing is crucial in ensuring that the final system performs accurately, efficiently, and reliably under different conditions. It also verifies whether the system meets the intended requirements.

## Test Plan

The test plan outlines the strategy for testing the core components of your number plate recognition system.

Test Scope

* Plate detection from images and video using YOLOv8.
* OCR recognition using EasyOCR (both English and Nepali plates).
* Real-time input handling via webcam or video stream.
* Accuracy, speed, and failure case monitoring.

Test Strategy

* Black-box testing for web UI and input/output handling.
* White-box testing for model performance and internal logic.
* Manual testing on a variety of images (clear, blurry, angled).
* Automation scripts to batch test recognition on test dataset.

Test Deliverables

* Test cases and result logs (in .txt or .csv format).
* Screenshots of successful and failed recognitions.
* Precision/recall charts for detection and OCR results.
* Final bug report and resolved issues list.

## Types and level of testing

1. Unit Testing

Each module tested in isolation:

* YOLO detection
* OCR reading
* Image pre-processing functions

Tools: Python unittest or manual assertions.

Example:

assert ocr("test\_plate.jpg") == "BA 5 CHA 1234"

1. Integration Testing

* Testing the combined performance of: Detection → OCR → Result Display → Output storage
* Focus: smooth data flow between modules.

Test Case:

Upload image → Check if CSV & image output are generated → Check correctness of recognized text.

1. System Testing

End-to-end testing of the full application using Flask UI:

* Upload image or video
* Detect plate
* Recognize characters
* Display + store results

Goal: Ensure the full system works across all routes:

* /image
* /video
* /realtime

1. Acceptance Testing

Conducted by supervisors or faculty:

* Verify system meets proposal requirements.
* Evaluate Devanagari and English recognition capability.
* Approve UI usability and result accuracy.

Example Checklist:

* Detects plate in real-time
* Recognizes English characters accurately
* Devanagari accuracy ~60% (in-progress)
* Saves outputs properly

Summary of Testing Phase

|  |  |  |
| --- | --- | --- |
| Test Type | Objective | Status |
| Unit Testing | Test model & functions individually | Completed |
| Integration Test | Test end-to-end logic connection | Completed |
| System Testing | Full system with UI | Completed |
| Acceptance Test | Supervisor validation | Ongoing |