Galaxy ALPR AI/ML Services Documentation

📋 Table of Contents

🚀 Quick Start

Installation

```
# Clone the repository
git clone <repository-url>
```

```
cd galaxy_alpr_core

# Install dependencies
pip install -r requirements.txt

# Install API-specific dependencies
pip install fastapi uvicorn python-multipart
```

Environment Setup

1. **Create `.env` file:**

   ```
   GEMINI_API_KEY=your_gemini_api_key_here
   ```

2. **Download AI models** and place in `models/` directory:

   - `model_vehicle_detector_yolo11n_v2.pt`
   - `model_plate_detector_yolo11n_v3.pt`

3. **Ensure directory structure:**

   ```
   galaxy_alpr_core/
   ├── API.py                    # Main API file
   ├── galaxy_alpr_core/         # Core ALPR modules
   ├── models/                   # AI model files
   ├── images_processed/         # Auto-created output directories
   └── .env                      # Environment variables
   ```

Running the API

```
# Start the API server
python API.py

# Or using uvicorn directly
uvicorn API:app --host 0.0.0.0 --port 8000 --reload
```

API Access:

- **Swagger Documentation**: http://localhost:8000/docs
- **ReDoc Documentation**: http://localhost:8000/redoc
- **Health Check**: http://localhost:8000/galaxy-alpr/v1/health

---

🏗 API Overview

Base URL Structure

All API endpoints follow the versioned structure:

```
http://localhost:8000/galaxy-alpr/v1/{endpoint}
```

Authentication

Currently, the API does not require authentication. All endpoints are publicly accessible.

Response Format

All endpoints return JSON responses with consistent structure:

```json
{
  "success": true,
  "message": "Operation completed successfully",
  "data": { /* Result data */ },
  "processing_time_ms": 1250,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

📚 API Endpoints

Information Endpoints

`GET /galaxy-alpr/v1`

Root endpoint with API information

Response:

```json
{
  "message": "Galaxy ALPR Core API",
  "description": "Advanced AI-Powered Automatic License Plate Recognition System",
  "version": "1.0.0",
  "developer": "@GalaxyDeveloper",
  "year": "2025",
  "documentation": "/docs",
  "health": "/galaxy-alpr/v1/health",
  "pipeline_info": "/galaxy-alpr/v1/pipeline/info"
}
```

`GET /galaxy-alpr/v1/health`

Health check endpoint

Response:

```
{
  "status": "healthy",
  "version": "1.0.0",
  "uptime": "2:30:45.123456",
  "models_loaded": {
    "vehicle_detector": true,
    "plate_detector": true,
    "gemini_ocr": true
  },
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Status Values:

- `healthy` - All models loaded and API functioning normally
- `degraded` - Some models missing or issues detected

`GET /galaxy-alpr/v1/pipeline/info`

Get detailed information about the ALPR pipeline

Response:

```
{
  "success": true,
  "pipeline_info": {
    "pipeline_name": "ALPR Core Pipeline",
    "description": "Complete Automatic License Plate Recognition system",
    "steps": [
      {
        "step": 1,
        "name": "Vehicle Detection",
        "description": "Detect vehicles in the input image",
        "component": "VehicleDetector"
      }
      // ... additional steps
    ],
    "input": "Image file path and optional timestamp",
    "output": "Final paired vehicle and plate detection/recognition results"
  },
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

ALPR Core Endpoints

`POST /galaxy-alpr/v1/alpr-core/process-single-image`

Process single image through complete ALPR pipeline

Parameters:

- `file` (required): Image file (JPG, JPEG, PNG)
- `custom_timestamp` (optional): Custom timestamp in format "YYYY-MM-DD_HH-MM-SS"

Example Request:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/alpr-core/process-single-image" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@sample_image.jpg" \
  -F "custom_timestamp=2025-06-06_14-30-15"
```

Response Structure:

```
{
  "success": true,
  "message": "ALPR processing completed successfully",
  "data": {
    "timestamp": "2025-06-06T14:30:15Z",
    "uploaded_image_path": "images_processed/uploaded_vehicle_images/2025-06-06_14-30-15_uplo
    "detected_vehicle_image_path": "images_processed/detected_vehicle_images/2025-06-06_14-30
    "detected_plate_image_path": "images_processed/detected_plate_images/2025-06-06_14-30-15_
    "processing_time": {
      "vehicle_detection_ms": 150,
      "plate_detection_ms": 120,
      "plate_recognition_ms": 800,
      "total_ms": 1070
    },
    "summary": {
      "total_detections": 2,
      "vehicles_with_plates": 1,
      "vehicles_without_plates": 0,
      "plates_without_vehicles": 1
    },
    "detections": [
      {
        "detection_id": 1,
        "detection_type": "vehicle_with_plate",
        "pairing_method": "containment",
        "pairing_confidence": 0.92,
        "vehicle": {
          "vehicle_index": 1,
          "vehicle_class": "car",
          "vehicle_confidence_score": 0.95,
          "vehicle_bounding_box": [100, 150, 400, 300],
          "vehicle_image_path": "images_processed/cropped_vehicle_images/2025-06-06_14-30-15_
        },
        "plate": {
          "plate_index": 1,
          "plate_class": "plate",
          "plate_confidence_score": 0.88,
          "plate_bounding_box": [180, 250, 280, 290],
          "plate_image_path": "images_processed/cropped_plate_images/2025-06-06_14-30-15_dete
          "plate_number": "B 1234 CD",
          "plate_date": "08/29",
          "plate_text_color": "black",
          "plate_background_color": "white",
          "plate_icon": "",
          "plate_icon_color": "",
          "plate_blue_strip": "no",
          "plate_type": "private",
          "confidence_score": 0.95,
          "plate_region_code": "B",
          "plate_region_name": "Jakarta/Metro Jaya"
        }
      }
    ]
  },
  "processing_time_ms": 1250,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`POST /galaxy-alpr/v1/alpr-core/process-multiple-images`

Process multiple images through complete ALPR pipeline

Parameters:

- `files` (required): Array of image files (max 10 files)
- `custom_timestamp` (optional): Custom timestamp prefix

Example Request:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/alpr-core/process-multiple-images" \
  -H "Content-Type: multipart/form-data" \
  -F "files=@image1.jpg" \
  -F "files=@image2.jpg" \
  -F "custom_timestamp=batch_2025-06-06"
```

Response Structure:

```
{
  "success": true,
  "message": "Batch processing completed for 2 images",
  "data": {
    "total_images": 2,
    "results": [
      {
        "batch_index": 1,
        "original_filename": "image1.jpg",
        // ... complete ALPR result for image1
      },
      {
        "batch_index": 2,
        "original_filename": "image2.jpg",
        // ... complete ALPR result for image2
      }
    ],
    "batch_summary": {
      "total_detections": 3,
      "vehicles_with_plates": 2,
      "vehicles_without_plates": 0,
      "plates_without_vehicles": 1
    }
  },
  "processing_time_ms": 2500,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Component-Specific Endpoints

`POST /galaxy-alpr/v1/vehicle/detect-vehicle`

Vehicle detection only

Parameters:

- `file` (required): Image file
- `custom_timestamp` (optional): Custom timestamp

Response:

```
{
  "success": true,
  "message": "Vehicle detection completed successfully",
  "data": {
    "uploaded_image_path": "path/to/uploaded_image.jpg",
    "detected_vehicle_image_path": "path/to/detected_image.jpg",
    "list_cropped_vehicle_image_paths": ["path/to/vehicle1.jpg", "path/to/vehicle2.jpg"],
    "list_class_vehicle": ["car", "motorcycle"],
    "list_bounding_box_vehicle": [[100, 150, 400, 300], [450, 200, 650, 350]],
    "list_confidence_score_vehicle": [0.95, 0.87],
    "vehicle_detection_processing_time": 0.15
  },
  "processing_time_ms": 180,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`POST /galaxy-alpr/v1/plate/detect-plate`

License plate detection only

Parameters:

- `file` (required): Image file
- `custom_timestamp` (optional): Custom timestamp

Response:

```json
{
  "success": true,
  "message": "Plate detection completed successfully",
  "data": {
    "uploaded_image_path": "path/to/uploaded_image.jpg",
    "detected_plate_image_path": "path/to/detected_image.jpg",
    "list_cropped_plate_image_paths": ["path/to/plate1.jpg"],
    "list_class_plate": ["plate"],
    "list_bounding_box_plate": [[180, 250, 280, 290]],
    "list_confidence_score_plate": [0.88],
    "plate_detection_processing_time": 0.12
  },
  "processing_time_ms": 150,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`POST /galaxy-alpr/v1/plate/ocr-plate`

OCR processing only

Parameters:

- `file` (required): Plate image file

Response:

```
{
  "success": true,
  "message": "Plate OCR completed successfully",
  "data": [
    {
      "plate_number": "B 1234 CD",
      "plate_date": "08/29",
      "plate_text_color": "black",
      "plate_background_color": "white",
      "plate_icon": "",
      "plate_icon_color": "",
      "plate_blue_strip": "no",
      "plate_type": "private",
      "confidence_score": 0.95
    }
  ],
  "processing_time_ms": 850,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`POST /galaxy-alpr/v1/plate/recognize-plate`

Complete plate recognition (OCR + regional mapping)

Parameters:

- `file` (required): Plate image file

Response:

```
{
  "success": true,
  "message": "Plate recognition completed successfully",
  "data": {
    "list_result_plate_recognized": [
      {
        "plate_number": "B 1234 CD",
        "plate_date": "08/29",
        "plate_text_color": "black",
        "plate_background_color": "white",
        "plate_icon": "",
        "plate_icon_color": "",
        "plate_blue_strip": "no",
        "plate_type": "private",
        "confidence_score": 0.95,
        "plate_region_code": "B",
        "plate_region_name": "Jakarta/Metro Jaya"
      }
    ],
    "plate_recognition_processing_time": 0.85
  },
  "processing_time_ms": 900,
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

System Management Endpoints

`GET /galaxy-alpr/v1/config`

Get current system configuration

Response:

```
{
  "success": true,
  "configuration": {
    "models": {
      "vehicle_detector": "model_vehicle_detector_yolo11n_v2.pt",
      "plate_detector": "model_plate_detector_yolo11n_v3.pt"
    },
    "ocr": {
      "provider": "gemini",
      "model_name": "gemini-2.0-flash-lite-001"
    },
    "detection": {
      "vehicle_confidence_threshold": 0.25,
      "plate_confidence_threshold": 0.25,
      "vehicle_padding": 25,
      "plate_padding": 25
    },
    "output": {
      "uploaded_vehicle_image_dir": "images_processed/uploaded_vehicle_images",
      "detected_vehicle_image_dir": "images_processed/detected_vehicle_images",
      "cropped_vehicle_image_dir": "images_processed/cropped_vehicle_images",
      "uploaded_plate_image_dir": "images_processed/uploaded_plate_images",
      "detected_plate_image_dir": "images_processed/detected_plate_images",
      "cropped_plate_image_dir": "images_processed/cropped_plate_images",
      "results_dir": "images_processed/results"
    },
    "image_formats": [".jpg", ".jpeg", ".png"],
    "timezone": "Asia/Makassar"
  },
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`GET /galaxy-alpr/v1/stats`

Get system statistics and status

Response:

```json
{
  "success": true,
  "statistics": {
    "system": {
      "uptime": "2:30:45.123456",
      "uptime_seconds": 9045.123456,
      "start_time": "2025-06-06T12:00:00.000000",
      "current_time": "2025-06-06T14:30:15.123456"
    },
    "models": {
      "vehicle_detector_loaded": true,
      "plate_detector_loaded": true,
      "gemini_api_configured": true
    },
    "configuration": {
      "vehicle_confidence_threshold": 0.25,
      "plate_confidence_threshold": 0.25,
      "supported_formats": [".jpg", ".jpeg", ".png"],
      "timezone": "Asia/Makassar"
    }
  },
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Utility Endpoints

`GET /galaxy-alpr/v1/images/{image_path}`

Retrieve processed images

Parameters:

- `image_path` (path): Relative path to the image file

Example:

```
GET /galaxy-alpr/v1/images/uploaded_vehicle_images/2025-06-06_14-30-15_uploaded_image.jpg
GET /galaxy-alpr/v1/images/cropped_plate_images/2025-06-06_14-30-15_detected_plate1.jpg
```

Response: Returns the image file directly

Testing Endpoints

`GET /galaxy-alpr/v1/test/connection`

Test API connection

Response:

```
{
  "status": "connected",
  "message": "Galaxy ALPR API is running",
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

`GET /galaxy-alpr/v1/test/models`

Test model loading and availability

Response:

```
{
  "success": true,
  "models": {
    "vehicle_detector": {
      "status": "loaded",
      "model_path": "/path/to/model_vehicle_detector_yolo11n_v2.pt",
      "classes": {
        "0.0": "car",
        "1.0": "motorcycle"
      }
    },
    "plate_detector": {
      "status": "loaded",
      "model_path": "/path/to/model_plate_detector_yolo11n_v3.pt",
      "classes": {
        "0.0": "plate"
      }
    },
    "gemini_ocr": {
      "status": "configured",
      "model_name": "gemini-2.0-flash-lite-001",
      "api_key_configured": true
    }
  },
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

🔧 Request/Response Models

Standard Response Models

ALPRResponse

```
{
  "success": bool,                      # Request success status
  "message": str,                       # Response message
  "data": Optional[Dict],             # ALPR processing results
  "processing_time_ms": Optional[int], # Total processing time in milliseconds
  "timestamp": str                    # Processing timestamp (ISO format)
}
```

HealthResponse

```
{
  "status": str,                      # API health status ("healthy"/"degraded")
  "version": str,                     # API version
  "uptime": str,                      # API uptime
  "models_loaded": Dict[str, bool],   # Model loading status
  "timestamp": str                    # Health check timestamp
}
```

Error Handling

ErrorResponse

```
{
  "success": false,
  "error": str,                       # Error message
  "error_type": str,                  # Type of error
  "timestamp": str                    # Error timestamp
}
```

Common HTTP Status Codes

- **200 OK** - Request successful
- **400 Bad Request** - Invalid request parameters or unsupported file format

- **404 Not Found** - Image file not found
- **500 Internal Server Error** - Server processing error

🧪 Usage Examples

Complete ALPR Processing

Single Image:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/alpr-core/process-single-image" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@car_with_plate.jpg"
```

Multiple Images:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/alpr-core/process-multiple-images" \
  -H "Content-Type: multipart/form-data" \
  -F "files=@image1.jpg" \
  -F "files=@image2.jpg" \
  -F "files=@image3.jpg"
```

Component-Level Processing

Vehicle Detection:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/vehicle/detect-vehicle" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@street_scene.jpg"
```

Plate Detection:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/plate/detect-plate" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@car_front.jpg"
```

Plate OCR:

```
curl -X POST "http://localhost:8000/galaxy-alpr/v1/plate/ocr-plate" \
  -H "Content-Type: multipart/form-data" \
  -F "file=@license_plate.jpg"
```

System Monitoring

Health Check:

```
curl -X GET "http://localhost:8000/galaxy-alpr/v1/health"
```

System Statistics:

```
curl -X GET "http://localhost:8000/galaxy-alpr/v1/stats"
```

Configuration:

```
curl -X GET "http://localhost:8000/galaxy-alpr/v1/config"
```

Python Client Example

```python
import requests
import json

# API base URL
BASE_URL = "http://localhost:8000/galaxy-alpr/v1"

def process_single_image(image_path):
    """Process a single image through ALPR pipeline"""
    url = f"{BASE_URL}/alpr-core/process-single-image"

    with open(image_path, 'rb') as f:
        files = {'file': f}
        response = requests.post(url, files=files)

    if response.status_code == 200:
        result = response.json()
        print(f"Success: {result['success']}")
        print(f"Processing time: {result['processing_time_ms']}ms")
        print(f"Detections: {len(result['data']['detections'])}")
        return result
    else:
        print(f"Error: {response.status_code}")
        print(response.json())
        return None

def check_health():
    """Check API health status"""
    url = f"{BASE_URL}/health"
    response = requests.get(url)

    if response.status_code == 200:
        health = response.json()
        print(f"Status: {health['status']}")
        print(f"Uptime: {health['uptime']}")
        print(f"Models loaded: {health['models_loaded']}")
        return health
    else:
        print(f"Health check failed: {response.status_code}")
        return None

# Usage
if __name__ == "__main__":
    # Check API health
    health_status = check_health()

    # Process an image
    if health_status and health_status['status'] == 'healthy':
        result = process_single_image("path/to/your/image.jpg")
        if result:
            # Process the results
            for detection in result['data']['detections']:
                if detection['detection_type'] == 'vehicle_with_plate':
                    vehicle = detection['vehicle']
                    plate = detection['plate']
                    print(f"Vehicle: {vehicle['vehicle_class']}")
                    print(f"Plate: {plate['plate_number']}")
                    print(f"Region: {plate['plate_region_name']}")
```

⚙️ Configuration

System Configuration

The API uses YAML configuration in `galaxy_alpr_core/config/config.yaml`:

```
models:
  vehicle_detector: models/model_vehicle_detector_yolo11n_v2.pt
```

```yaml
  plate_detector: models/model_plate_detector_yolo11n_v3.pt

ocr:
  provider: gemini
  model_name: gemini-2.0-flash-lite-001

detection:
  vehicle_confidence_threshold: 0.25
  plate_confidence_threshold: 0.25
  vehicle_padding: 25
  plate_padding: 25

output:
  save_detected_images: true
  uploaded_vehicle_image_dir: images_processed/uploaded_vehicle_images
  detected_vehicle_image_dir: images_processed/detected_vehicle_images
  cropped_vehicle_image_dir: images_processed/cropped_vehicle_images
  uploaded_plate_image_dir: images_processed/uploaded_plate_images
  detected_plate_image_dir: images_processed/detected_plate_images
  cropped_plate_image_dir: images_processed/cropped_plate_images
  results_dir: images_processed/results

image_formats:
  - .jpg
  - .jpeg
  - .png

timezone: Asia/Makassar
```

File Upload Limits

- **Maximum file size**: Determined by FastAPI/uvicorn settings
- **Supported formats**: JPG, JPEG, PNG
- **Batch processing limit**: Maximum 10 files per request

- **Temporary file cleanup**: Automatic cleanup via background tasks

---

🚨 Error Codes & Troubleshooting

Common Error Codes

400 Bad Request

```json
{
  "success": false,
  "error": "Unsupported file format. Allowed formats: ['.jpg', '.jpeg', '.png']",
  "error_type": "HTTPException",
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Causes:

- Unsupported file format
- Invalid custom timestamp format
- Too many files in batch processing (>10)

404 Not Found

```json
{
  "success": false,
  "error": "Image not found",
  "error_type": "HTTPException",
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Causes:

- Requested image file doesn't exist
- Invalid image path

500 Internal Server Error

```json
{
  "success": false,
  "error": "ALPR processing failed: Model not found",
  "error_type": "HTTPException",
  "timestamp": "2025-06-06T14:30:15.123456"
}
```

Causes:

- Missing AI model files
- Invalid GEMINI_API_KEY
- Internal processing errors
- Insufficient system resources

Troubleshooting Guide

Model Loading Issues

Problem: Vehicle/Plate detector not loading

```json
{
  "vehicle_detector": false,
  "plate_detector": false
}
```

Solution:

1. Verify model files exist in `models/` directory
2. Check file permissions
3. Ensure correct model file names in config

Gemini API Issues

Problem: OCR processing failures

```
{
  "gemini_ocr": false
}
```

Solution:

1. Verify `GEMINI_API_KEY` in `.env` file
2. Check API key validity
3. Verify internet connection
4. Check Gemini API rate limits

File Upload Issues

Problem: File upload failures

Solution:

1. Check file format (must be JPG, JPEG, or PNG)

2. Verify file size limits
3. Ensure proper multipart/form-data encoding
4. Check disk space for temporary files

Performance Issues

Problem: Slow processing times

Solution:

1. Use GPU acceleration for YOLO models
2. Reduce image resolution
3. Increase confidence thresholds
4. Monitor system resources

📊 Performance & Limitations

Processing Times

Component	CPU Only	GPU (RTX 3060)	GPU (RTX 4090)
Vehicle Detection	800-1500ms	50-150ms	20-80ms
Plate Detection	600-1200ms	40-120ms	15-60ms
OCR Recognition	800-1500ms	800-1500ms	800-1500ms
Total	**2.2-4.2s**	**0.9-1.8s**	**0.8-1.6s**

Note: OCR time is network-dependent due to Gemini API calls

Rate Limits

Gemini API Limits (Free Tier):

- **Requests per minute**: 30
- **Tokens per minute**: 1,000,000
- **Requests per day**: 1,500

API Server Limits:

- **Batch processing**: Maximum 10 files per request
- **Concurrent requests**: Limited by server resources
- **File upload timeout**: 60 seconds (configurable)

File Size Limits

- **Maximum image size**: 20MB (recommended: <5MB for faster processing)
- **Minimum image resolution**: 224x224 pixels
- **Recommended resolution**: 640x480 to 1920x1080 pixels

System Requirements

Minimum Requirements:

- **RAM**: 8GB
- **Storage**: 10GB free space
- **Python**: 3.8+
- **Internet**: Stable connection for Gemini API

Recommended Requirements:

- **RAM**: 16GB+
- **GPU**: NVIDIA GPU with 6GB+ VRAM
- **Storage**: 50GB+ free space
- **CPU**: 8+ cores

🏷️ Citation

If you use Galaxy ALPR API in your projects, please cite:

```
Galaxy ALPR Core API - Advanced AI-Powered License Plate Recognition System
Developed by @GalaxyDeveloper (2025)
Built with FastAPI, YOLOv11n, Google Gemini AI, and intelligent pairing algorithms
```

Galaxy ALPR Core API - Advanced AI-Powered License Plate Recognition System
Built with FastAPI, YOLOv11n, Google Gemini AI, and intelligent pairing algorithms

Developed by @GalaxyDeveloper - 2025