POLYTECH
Peter the Great
St.Petersburg Polytechnic
University

Recognition of potholes on a road

Course Work Report:
Corporate Information Systems

Prepared by:

Alexander Syrcev
Alexei Antonov
Dmitrii Panfilov
Ekaterina Shirimetova
Mikhail Rozhdestvenskiy
Nikolai Glebov

Submitted to: Potekhin V.V.

Saint Petersburg, 2023

Table of contents

01

Introduction

Information about our project, Goal, Tasks, Dataset

02

Diagrams

IDEF0, Swimlane and Use Case diagrams

03

Knowledge management

Knowledge base, Rule base, SPARQL

04

Technologies used

Backend and frontend, Demonstration

05

Results achieved

Examples of images and KPI's

06

Conclusion

Plans for future and Summarizing

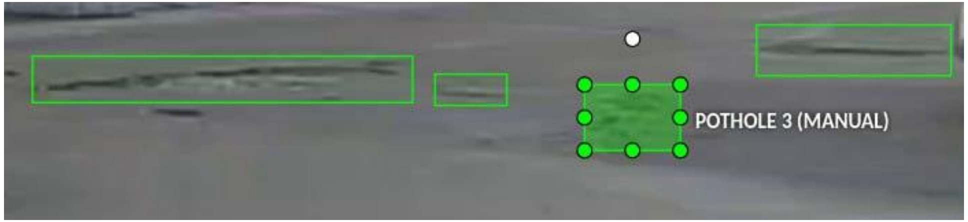
Project Goal

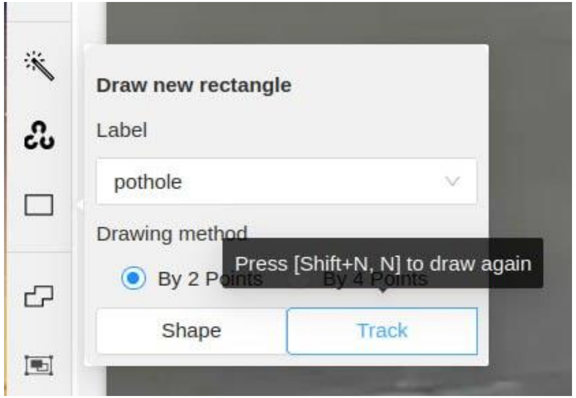
To develop a multi-agents system capable of detecting potholes in real time with a high degree of accuracy and with a sufficiently high processing speed, marking detected potholes as dangerous or relatively safe by their width in the image

Project Tasks

#1	Preparing a dataset	Looking for a dataset and labeling images
#2	Train the YOLOv8 model	Photos and Videos
#3	Develop a Knowledge Base	Knowledge base with consistency checking
#4	Develop API and clients	FastAPI backend + Telegram and Web App
#5	Integration and testing	FastAPI backend + Telegram and Web App

Dataset and Images labeling





Draw new rectangle

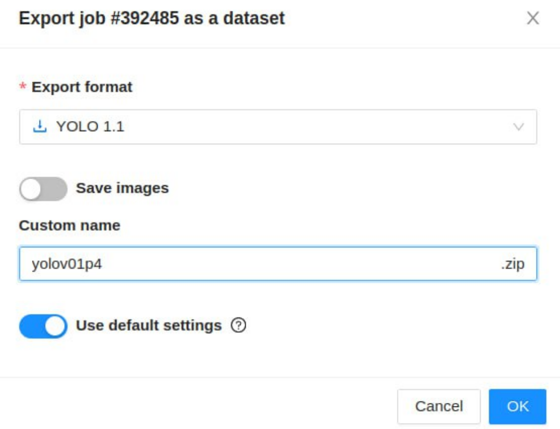
Label

pothole

Drawing method

● By 2 Points ○ By 4 Points

Press [Shift+N, N] to draw again



Export job #392485 as a dataset

* **Export format**

YOLO 1.1

Save images

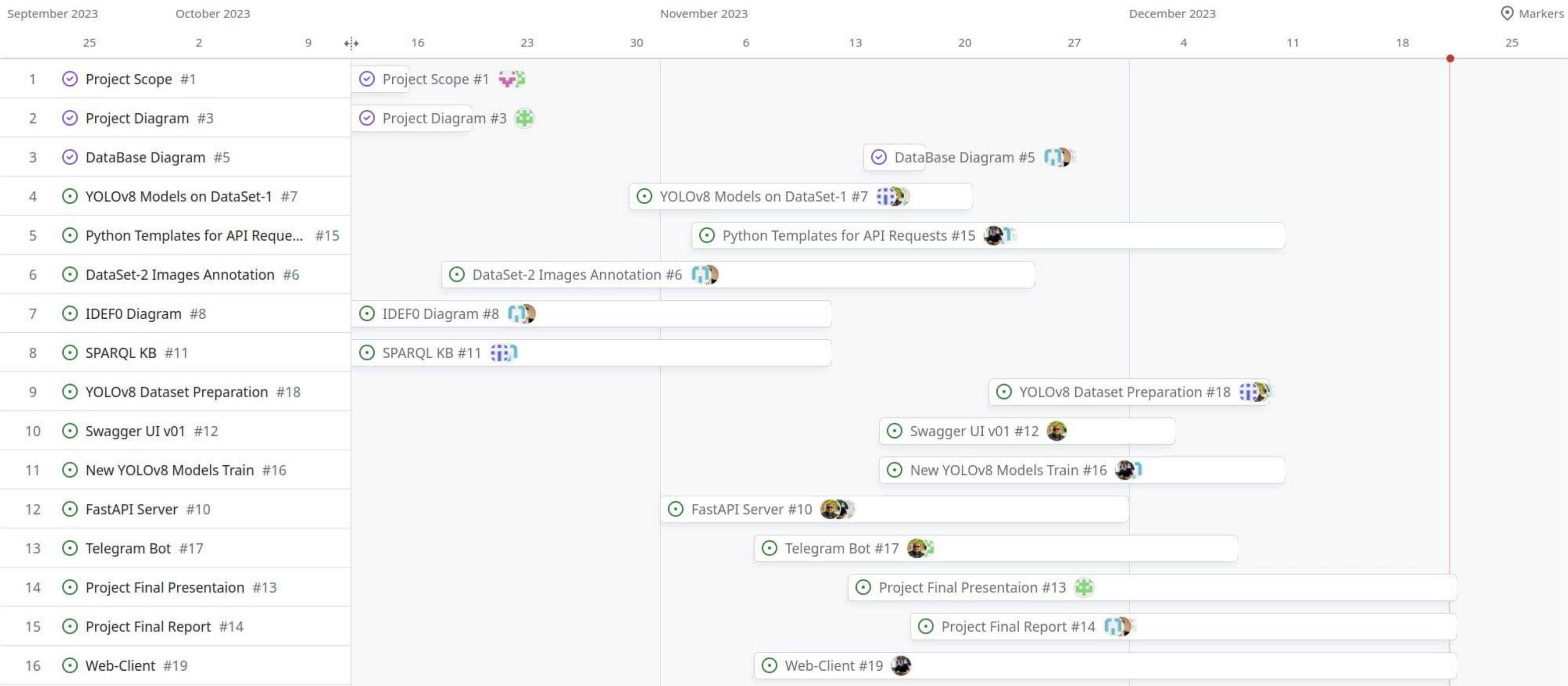
Custom name

yolov01p4 .zip

Use default settings

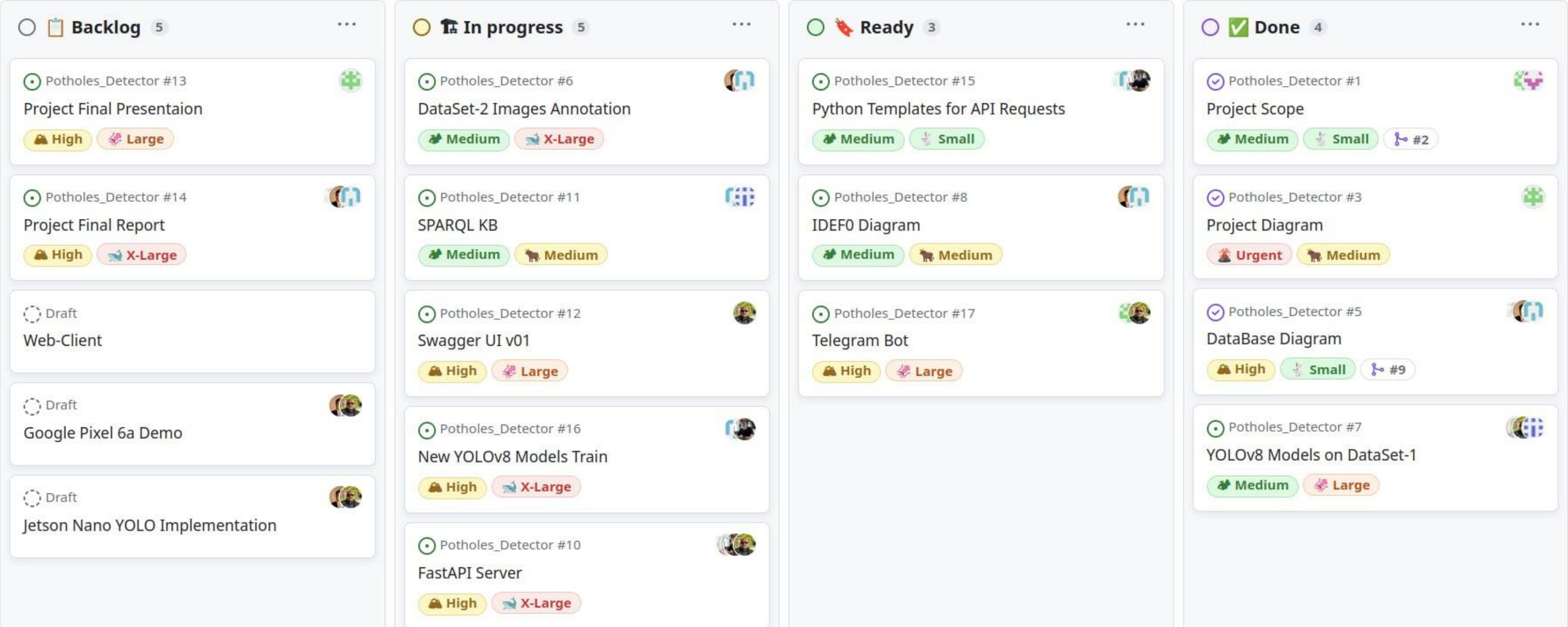
Project Diagrams

Gantt Chart



1	✓	Project Scope #1
2	✓	Project Diagram #3
3	✓	DataBase Diagram #5
4	⊙	YOLOv8 Models on DataSet-1 #7
5	⊙	Python Templates for API Reque... #15
6	⊙	DataSet-2 Images Annotation #6
7	⊙	IDEF0 Diagram #8
8	⊙	SPARQL KB #11
9	⊙	YOLOv8 Dataset Preparation #18
10	⊙	Swagger UI v01 #12
11	⊙	New YOLOv8 Models Train #16
12	⊙	FastAPI Server #10
13	⊙	Telegram Bot #17
14	⊙	Project Final Presentaion #13
15	⊙	Project Final Report #14
16	⊙	Web-Client #19

KANBAN Board



📋 Backlog 5

Potholes_Detector #13
Project Final Presentaion
High · Large

Potholes_Detector #14
Project Final Report
High · X-Large

Draft
Web-Client

Draft
Google Pixel 6a Demo

Draft
Jetson Nano YOLO Implementation

🏗 In progress 5

Potholes_Detector #6
DataSet-2 Images Annotation
Medium · X-Large

Potholes_Detector #11
SPARQL KB
Medium · Medium

Potholes_Detector #12
Swagger UI v01
High · Large

Potholes_Detector #16
New YOLOv8 Models Train
High · X-Large

Potholes_Detector #10
FastAPI Server
High · X-Large

🔖 Ready 3

Potholes_Detector #15
Python Templates for API Requests
Medium · Small

Potholes_Detector #8
IDEF0 Diagram
Medium · Medium

Potholes_Detector #17
Telegram Bot
High · Large

✅ Done 4

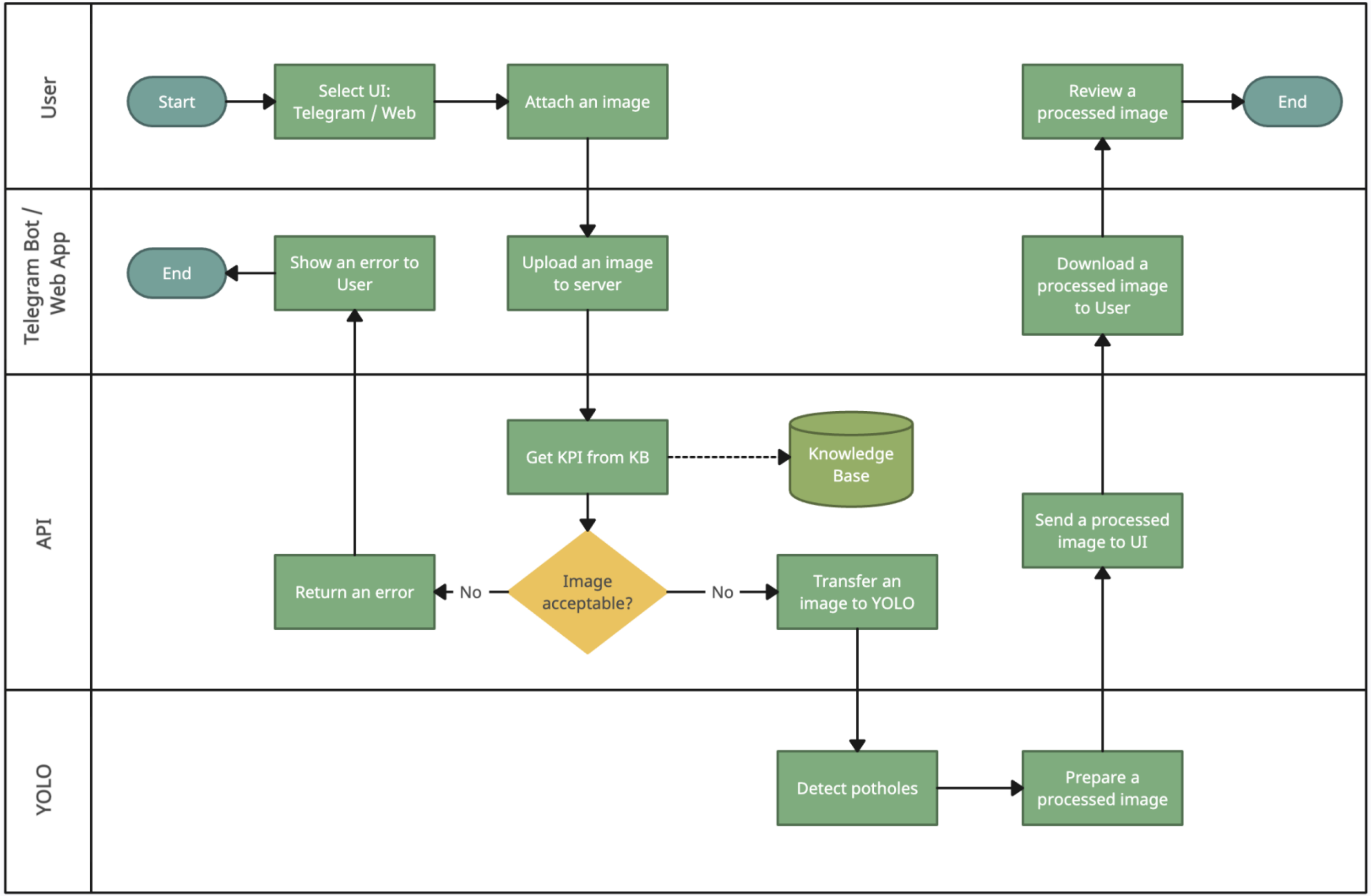
Potholes_Detector #1
Project Scope
Medium · Small · #2

Potholes_Detector #3
Project Diagram
Urgent · Medium

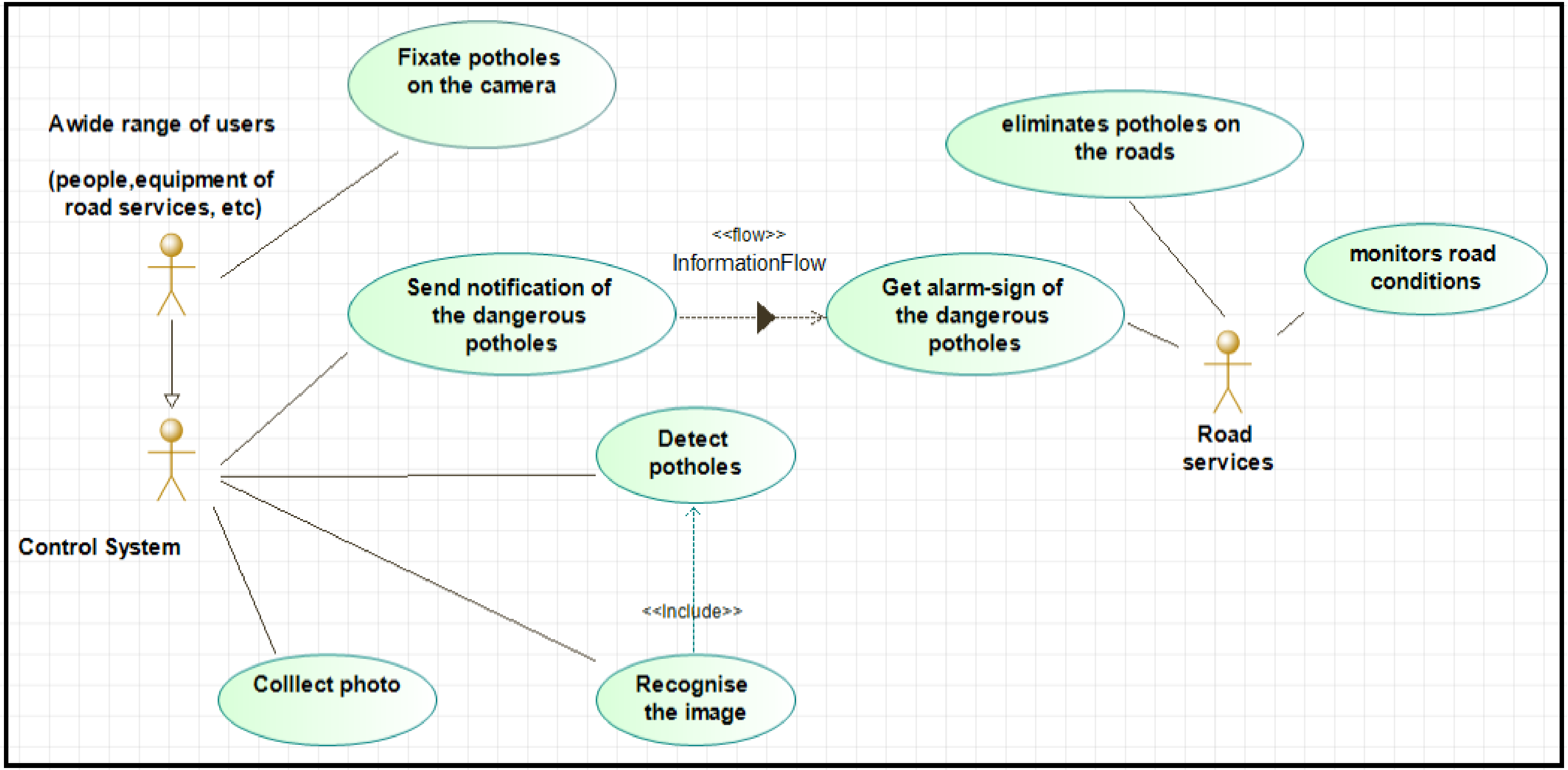
Potholes_Detector #5
DataBase Diagram
High · Small · #9

Potholes_Detector #7
YOLOv8 Models on DataSet-1
Medium · Large

Swimlane Diagram

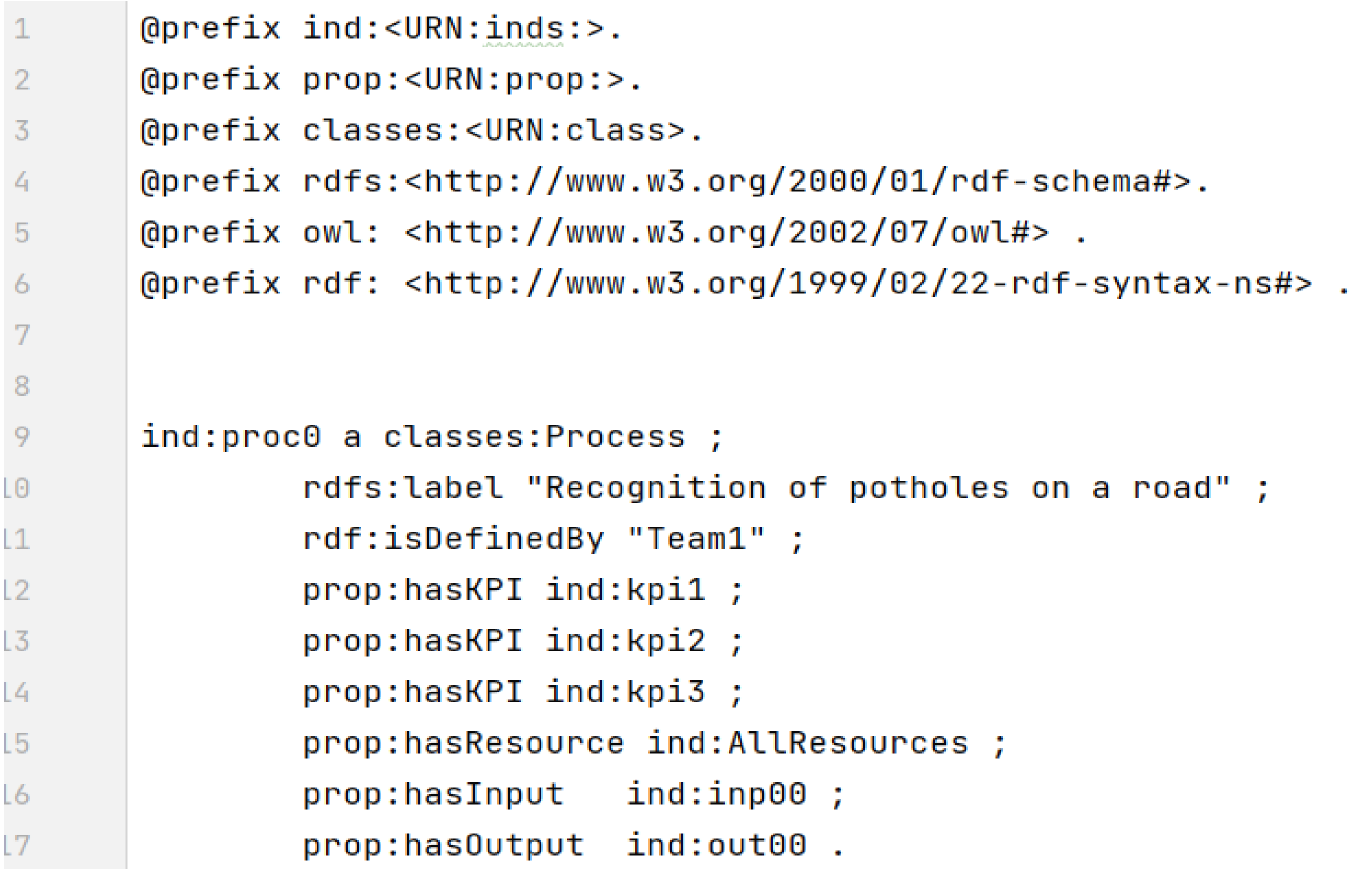

User
Telegram Bot / Web App
API
YOLO
Start
Select UI: Telegram / Web
Attach an image
Review a processed image
End
End
Show an error to User
Upload an image to server
Download a processed image to User
Get KPI from KB
Knowledge Base
Send a processed image to UI
Return an error
No
Image acceptable?
No
Transfer an image to YOLO
Detect potholes
Prepare a processed image

Use Case Diagram


Fixate potholes on the camera
Awide range of users
(people,equipment of road services, etc)
eliminates potholes on the roads
<<flow>>
InformationFlow
monitors road conditions
Send notification of the dangerous potholes
Get alarm-sign of the dangerous potholes
Road services
Detect potholes
Control System
<<Include>>
Colllect photo
Recognise the image

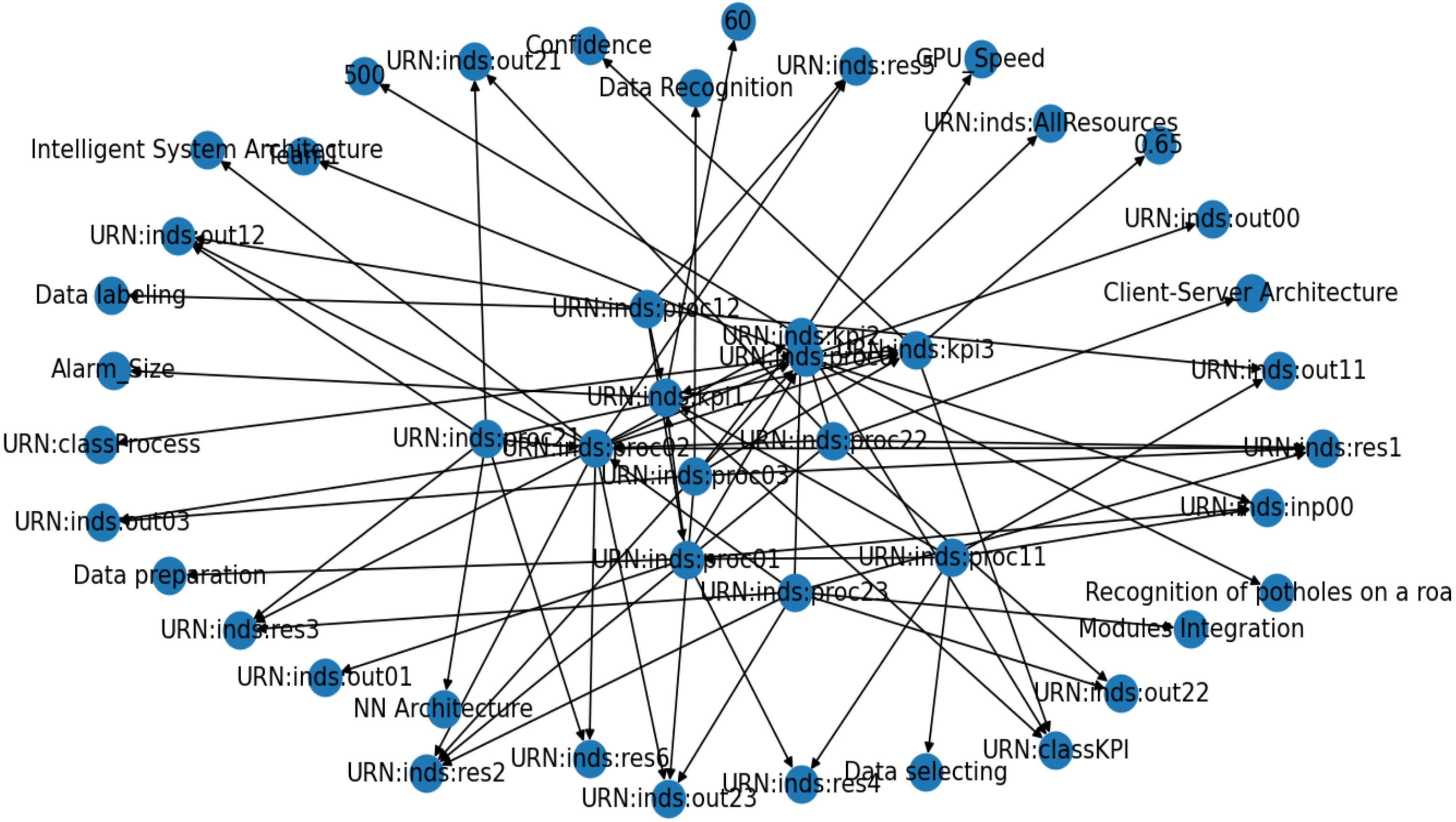
Knowledge Base
Rule Base
SPARQL

KB



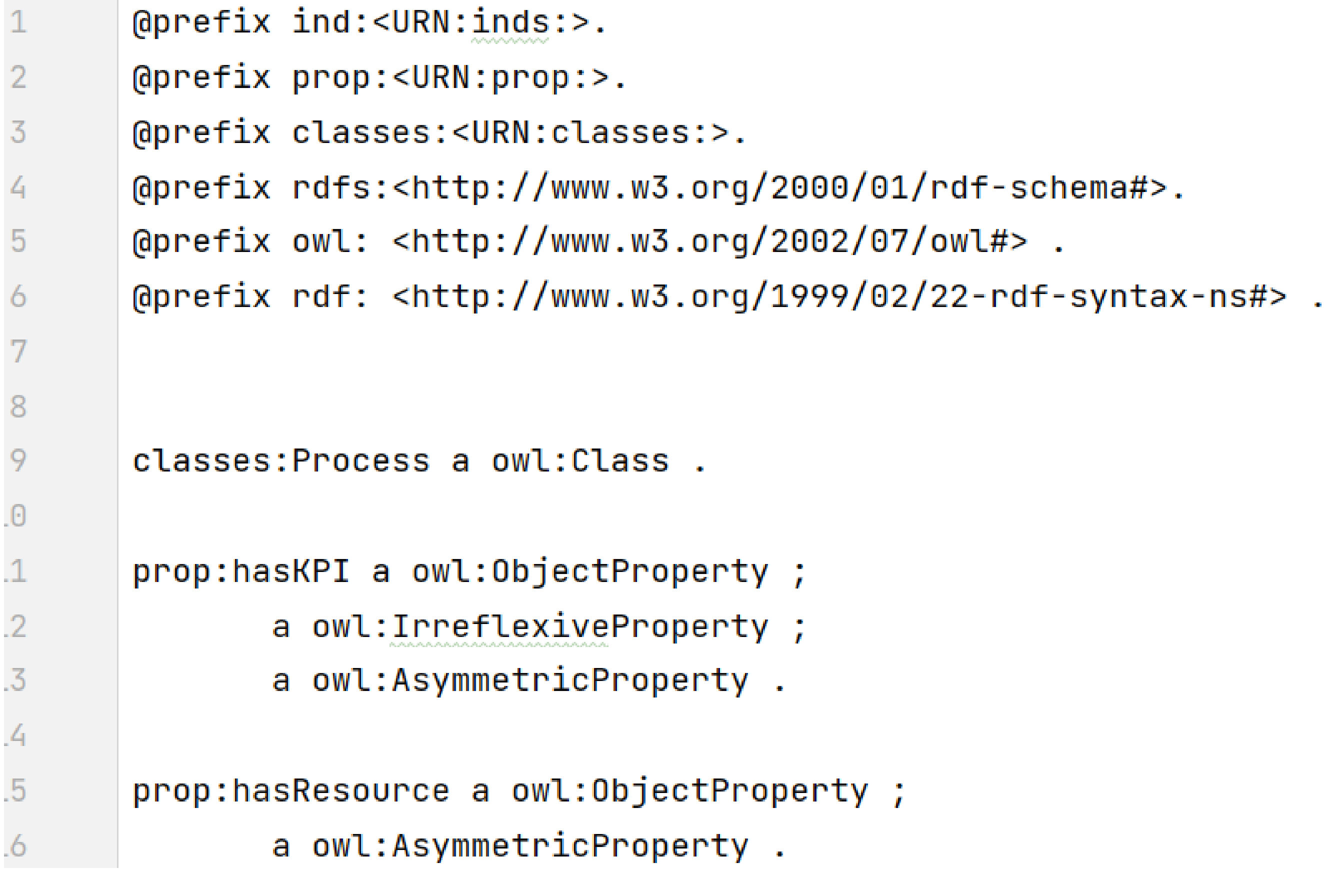
```
@prefix ind:<URN:inds:>.
@prefix prop:<URN:prop:>.
@prefix classes:<URN:class>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .


ind:proc0 a classes:Process ;
        rdfs:label "Recognition of potholes on a road" ;
        rdf:isDefinedBy "Team1" ;
        prop:hasKPI ind:kpi1 ;
        prop:hasKPI ind:kpi2 ;
        prop:hasKPI ind:kpi3 ;
        prop:hasResource ind:AllResources ;
        prop:hasInput   ind:inp00 ;
        prop:hasOutput  ind:out00 .
```

KB



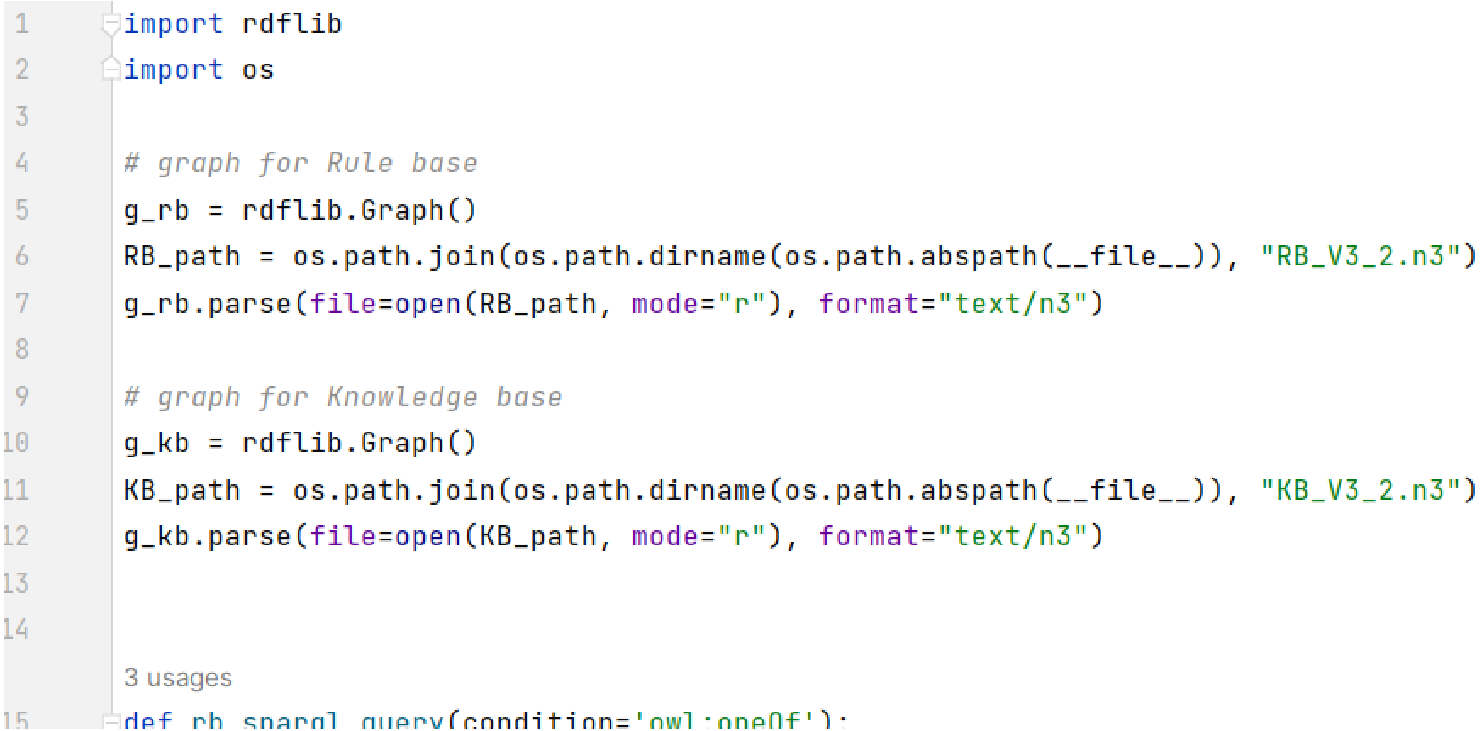
60
Confidence
500
URN:inds:out21
URN:inds:res5
GPU_Speed
Data Recognition
URN:inds:AllResources
0.65
Intelligent System Architecture
Team1
URN:inds:out00
URN:inds:out12
Client-Server Architecture
Data labeling
URN:inds:proc12
URN:inds:kpi2
URN:inds:proc00
URN:inds:kpi3
URN:inds:out11
Alarm_Size
URN:inds:kpi1
URN:classProcess
URN:inds:proc21
URN:inds:proc02
URN:inds:proc22
URN:inds:res1
URN:inds:proc03
URN:inds:inp00
URN:inds:out03
URN:inds:proc01
URN:inds:proc11
Data preparation
URN:inds:proc23
Recognition of potholes on a roa
URN:inds:res3
Modules Integration
URN:inds:out01
NN Architecture
URN:inds:out22
URN:classKPI
URN:inds:res6
URN:inds:res2
Data selecting
URN:inds:res4
URN:inds:out23

RB



```
@prefix ind:<URN:inds:>.
@prefix prop:<URN:prop:>.
@prefix classes:<URN:classes:>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .


classes:Process a owl:Class .

prop:hasKPI a owl:ObjectProperty ;
       a owl:IrreflexiveProperty ;
       a owl:AsymmetricProperty .

prop:hasResource a owl:ObjectProperty ;
       a owl:AsymmetricProperty .
```

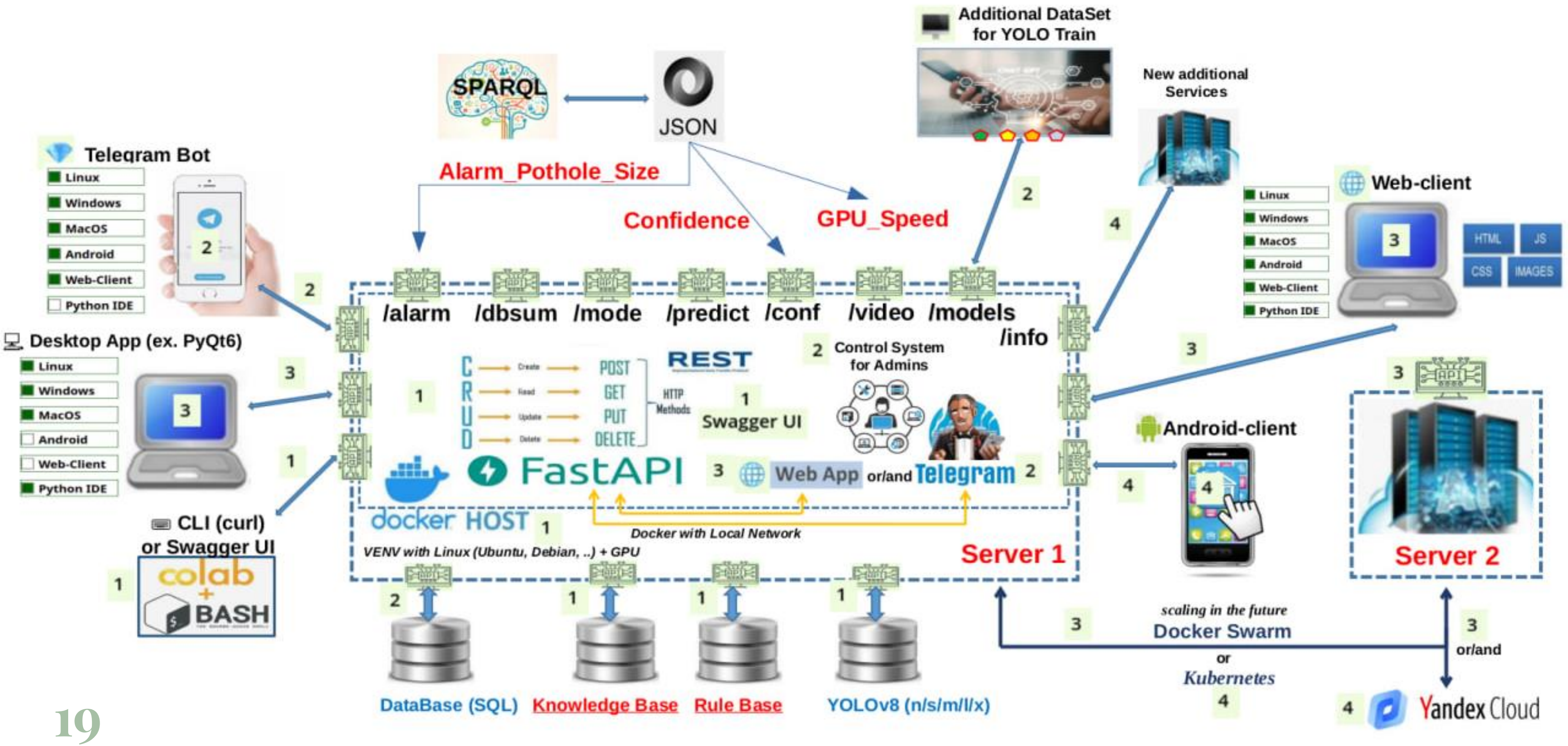
SPARQ



```python
import rdflib
import os

# graph for Rule base
g_rb = rdflib.Graph()
RB_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), "RB_V3_2.n3")
g_rb.parse(file=open(RB_path, mode="r"), format="text/n3")

# graph for Knowledge base
g_kb = rdflib.Graph()
KB_path = os.path.join(os.path.dirname(os.path.abspath(__file__)), "KB_V3_2.n3")
g_kb.parse(file=open(KB_path, mode="r"), format="text/n3")


3 usages
def rb_sparql_query(condition='owl:oneOf'):
```

Key Technologies

Multi-agent API Solution



Telegram Bot

- ■ Linux
- ■ Windows
- ■ MacOS
- ■ Android
- ■ Web-Client
- □ Python IDE

2

2

Desktop App (ex. PyQt6)

- ■ Linux
- ■ Windows
- ■ MacOS
- □ Android
- □ Web-Client
- ■ Python IDE

3

3

1

CLI (curl) or Swagger UI

1

colab + BASH

SPARQL

JSON

Alarm_Pothole_Size

Confidence

GPU_Speed

Additional DataSet for YOLO Train

2

New additional Services

4

/alarm /dbsum /mode /predict /conf /video /models /info

1

C Create POST
R Read GET
U Update PUT
D Delete DELETE

HTTP Methods

REST

1 Swagger UI

2 Control System for Admins

FastAPI

3 Web App or/and Telegram 2

docker HOST 1

Docker with Local Network

VENV with Linux (Ubuntu, Debian, ..) + GPU

Server 1

2 DataBase (SQL)

1 Knowledge Base

1 Rule Base

1 YOLOv8 (n/s/m/l/x)

Web-client

- ■ Linux
- ■ Windows
- ■ MacOS
- ■ Android
- ■ Web-Client
- ■ Python IDE

3

HTML JS CSS IMAGES

3

Android-client

4

4

3

Server 2

3

scaling in the future

Docker Swarm

or

Kubernetes

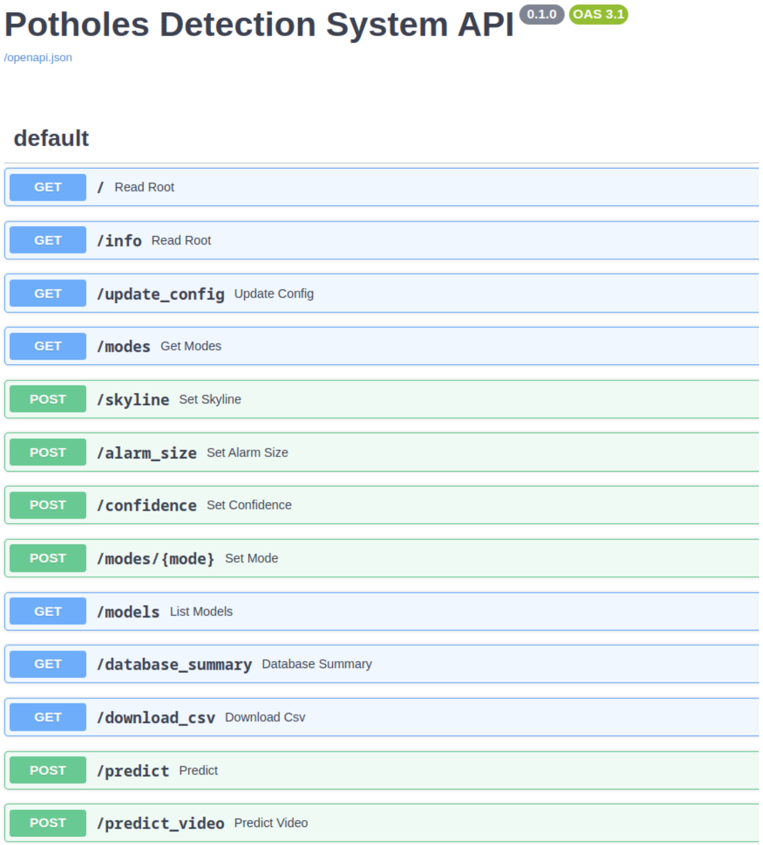
4

3

or/and

4 Yandex Cloud

FastAPI and Swagger

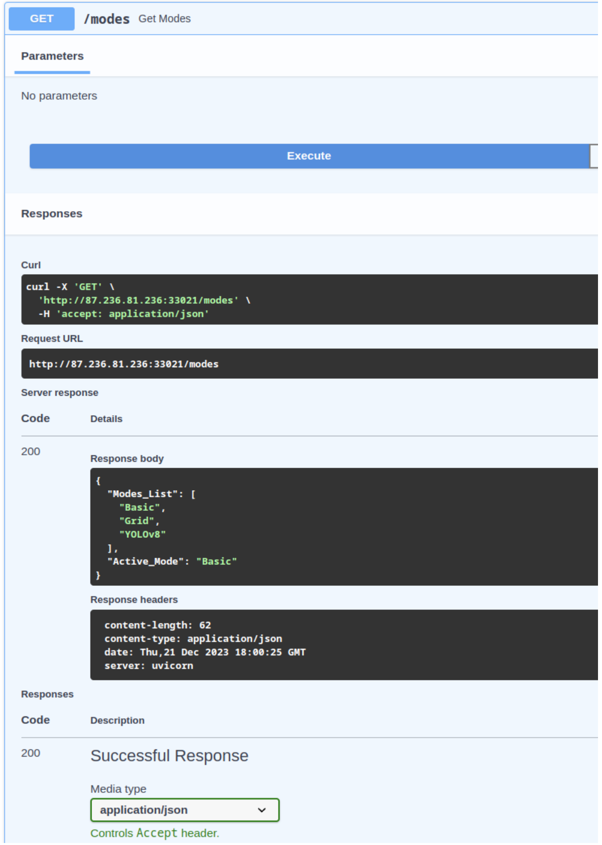


Potholes Detection System API 0.1.0 OAS 3.1

/openapi.json

default

- GET / Read Root
- GET /info Read Root
- GET /update_config Update Config
- GET /modes Get Modes
- POST /skyline Set Skyline
- POST /alarm_size Set Alarm Size
- POST /confidence Set Confidence
- POST /modes/{mode} Set Mode
- GET /models List Models
- GET /database_summary Database Summary
- GET /download_csv Download Csv
- POST /predict Predict
- POST /predict_video Predict Video



GET /modes Get Modes

Parameters

No parameters

Execute

Responses

Curl

```
curl -X 'GET' \
  'http://87.236.81.236:33021/modes' \
  -H 'accept: application/json'
```

Request URL

```
http://87.236.81.236:33021/modes
```

Server response

Code	Details
200	Response body

```
{
  "Modes_List": [
    "Basic",
    "Grid",
    "YOLOv8"
  ],
  "Active_Mode": "Basic"
}
```

Response headers

```
content-length: 62
content-type: application/json
date: Thu,21 Dec 2023 18:00:25 GMT
server: uvicorn
```

Responses

Code	Description
200	Successful Response

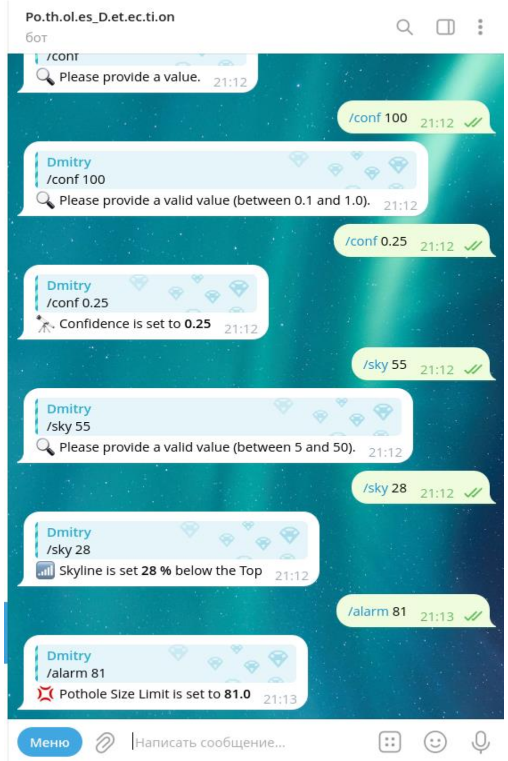
Media type

application/json

Controls Accept header.

User Interfaces

UI: Telegram Bot



/conf
Please provide a value. 21:12

/conf 100 21:12

Dmitry
/conf 100
Please provide a valid value (between 0.1 and 1.0). 21:12

/conf 0.25 21:12

Dmitry
/conf 0.25
Confidence is set to **0.25** 21:12

/sky 55 21:12

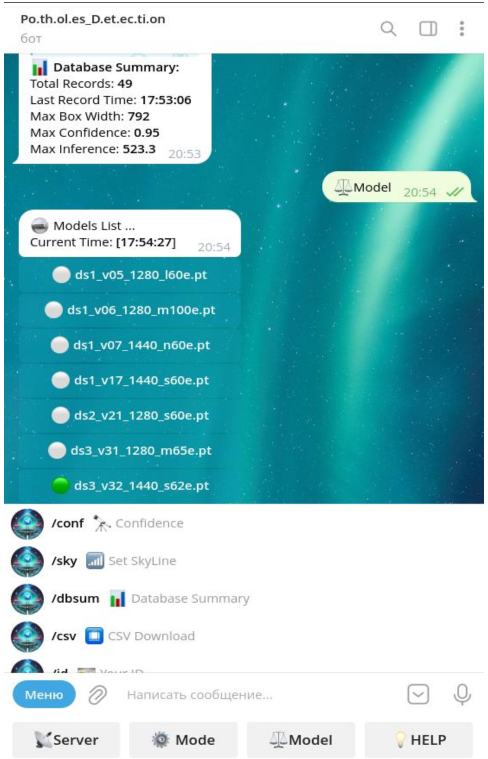
Dmitry
/sky 55
Please provide a valid value (between 5 and 50). 21:12

/sky 28 21:12

Dmitry
/sky 28
Skyline is set **28 %** below the Top 21:12

/alarm 81 21:13

Dmitry
/alarm 81
Pothole Size Limit is set to **81.0** 21:13

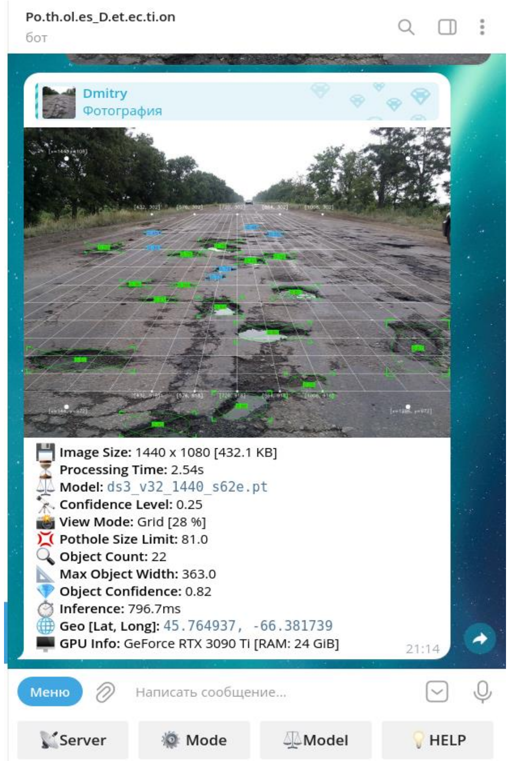


Database Summary:
Total Records: **49**
Last Record Time: **17:53:06**
Max Box Width: **792**
Max Confidence: **0.95**
Max Inference: **523.3** 20:53

Model 20:54

Models List ...
Current Time: **[17:54:27]** 20:54

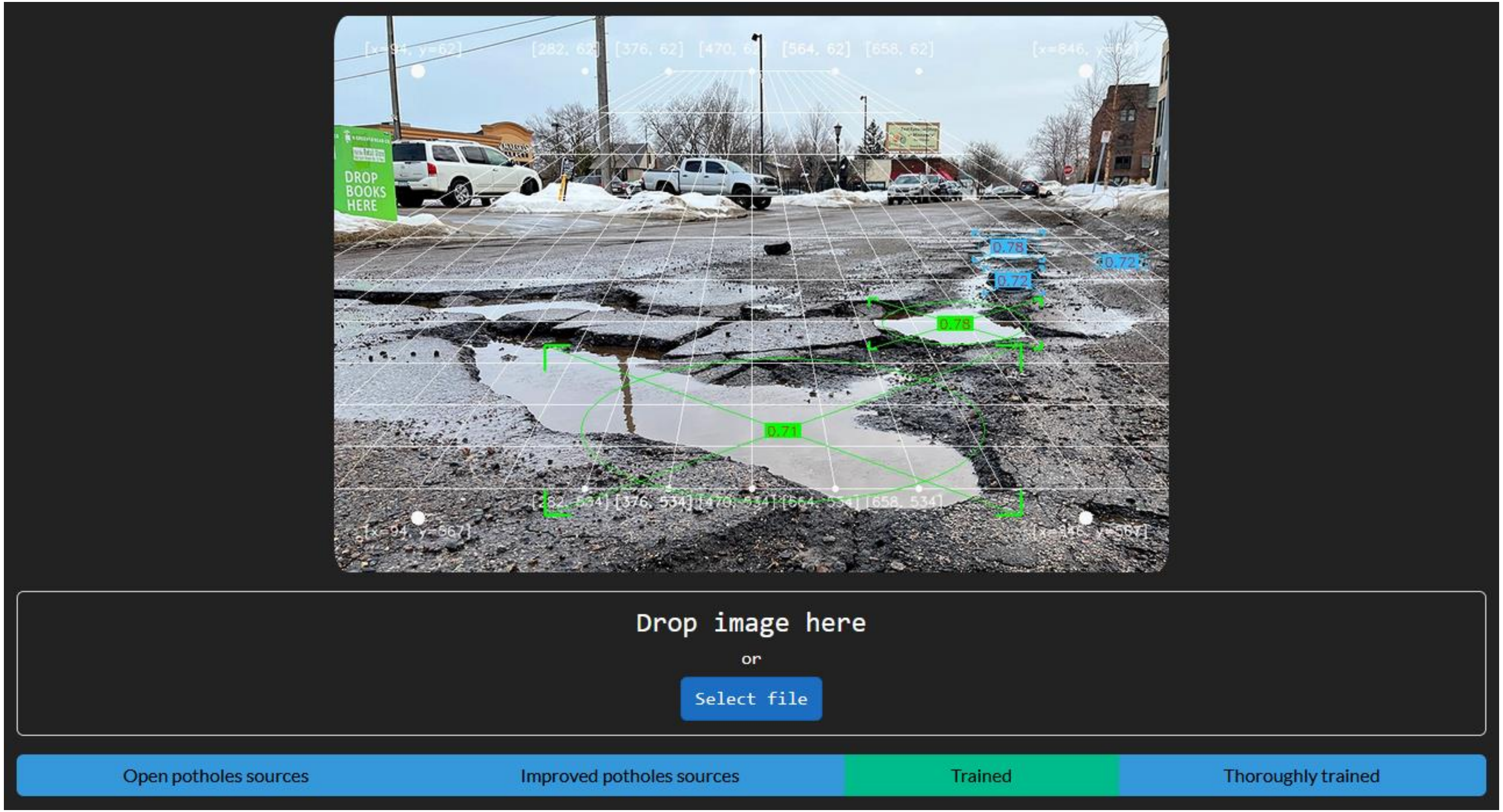
○ ds1_v05_1280_l60e.pt
○ ds1_v06_1280_m100e.pt
○ ds1_v07_1440_n60e.pt
○ ds1_v17_1440_s60e.pt
○ ds2_v21_1280_s60e.pt
○ ds3_v31_1280_m65e.pt
● ds3_v32_1440_s62e.pt



Dmitry
Фотография

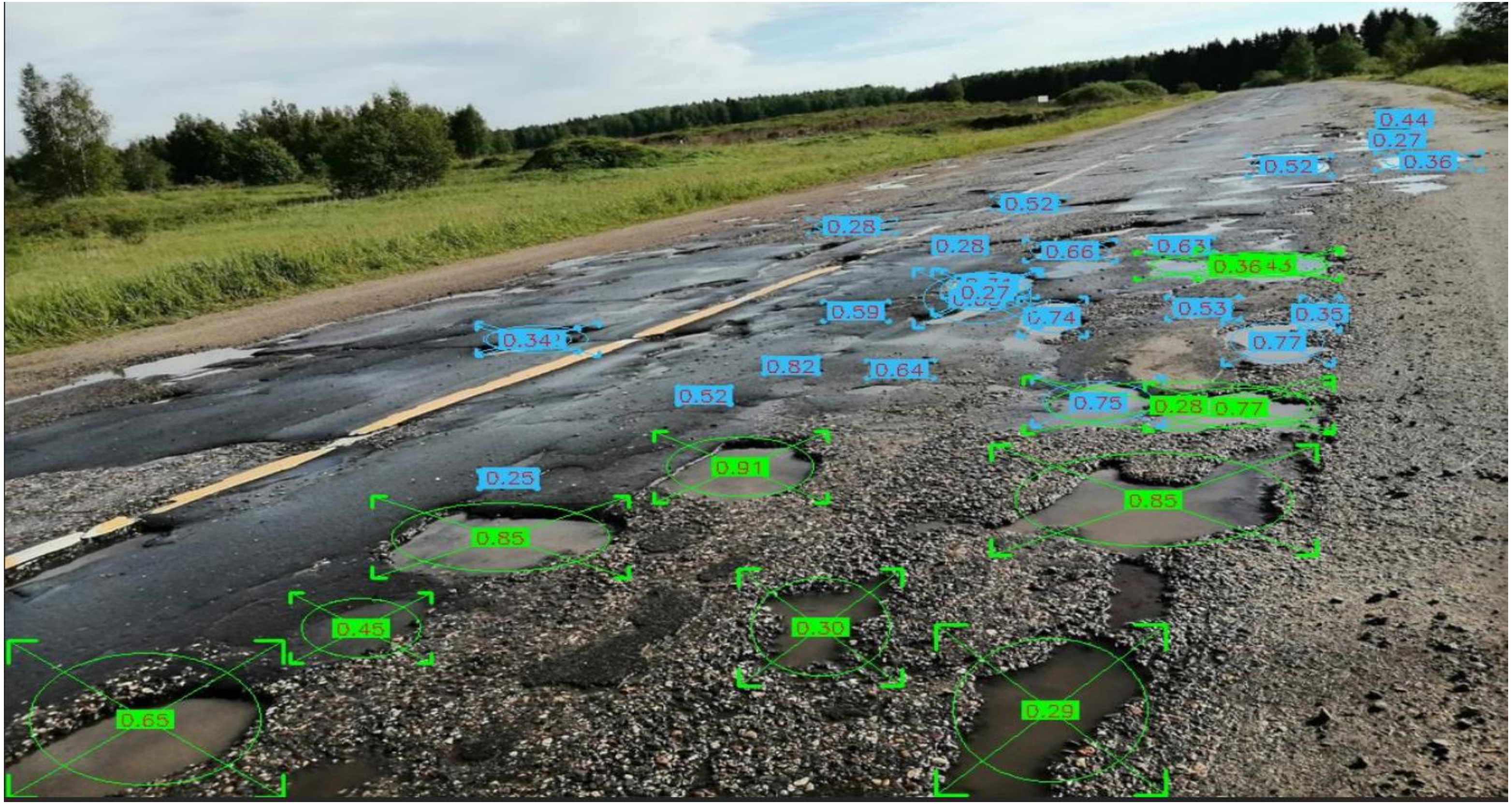
Image Size: 1440 x 1080 [432.1 KB]
Processing Time: 2.54s
Model: `ds3_v32_1440_s62e.pt`
Confidence Level: 0.25
View Mode: Grid [28 %]
Pothole Size Limit: 81.0
Object Count: 22
Max Object Width: 363.0
Object Confidence: 0.82
Inference: 796.7ms
Geo [Lat, Long]: `45.764937, -66.381739`
GPU Info: GeForce RTX 3090 Ti [RAM: 24 GiB] 21:14

UI: Web Application



Examples of Output

Example #1


0.44
0.27
0.36
0.52
0.52
0.28
0.28
0.66
0.63
0.36
0.27
0.59
0.74
0.53
0.35
0.34
0.77
0.82
0.64
0.52
0.75
0.28
0.77
0.25
0.91
0.85
0.85
0.45
0.30
0.65
0.29

Example #2

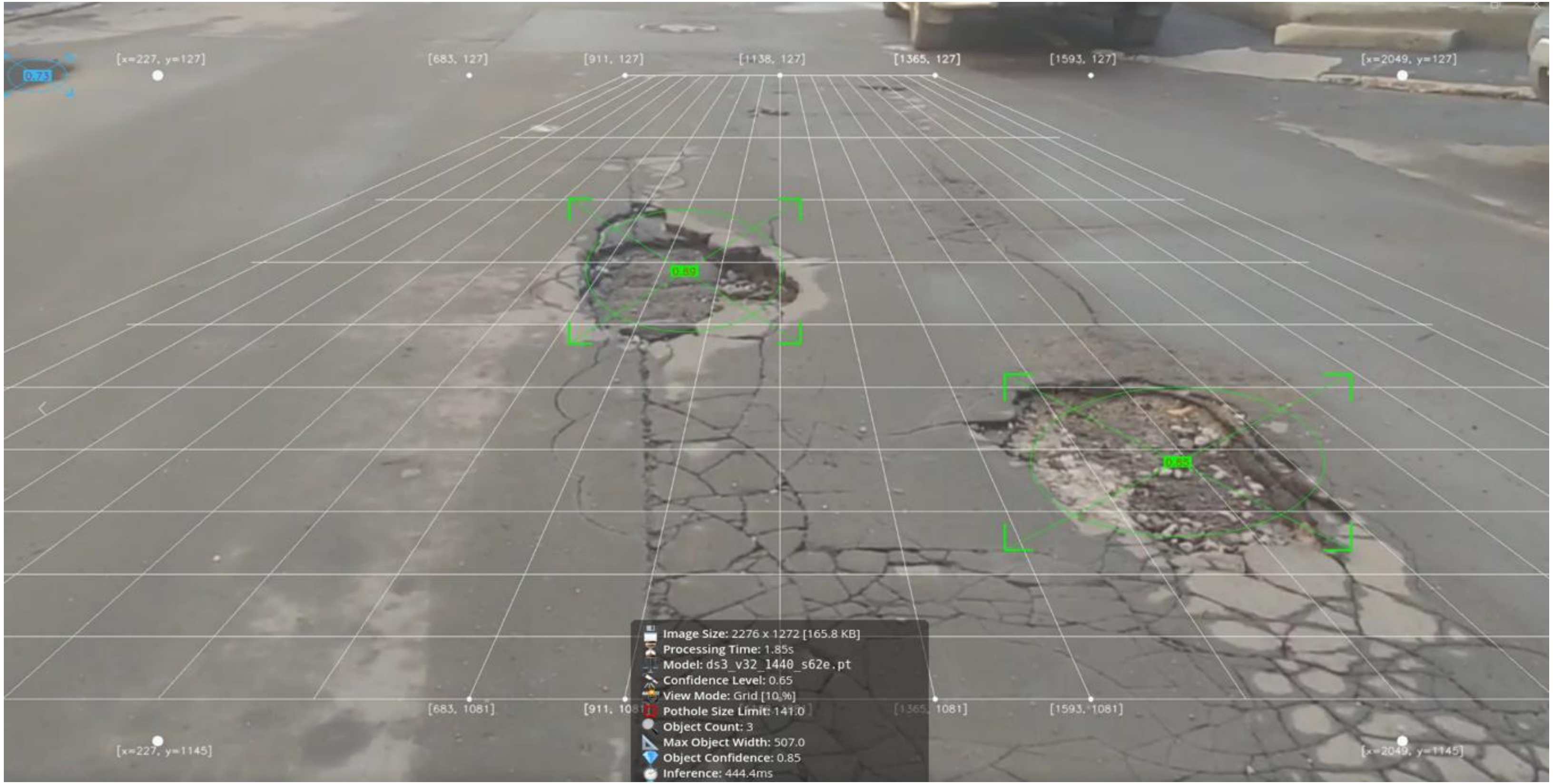


Image Size: 2276 x 1272 [165.8 KB]
Processing Time: 1.85s
Model: ds3_v32_1440_s62e.pt
Confidence Level: 0.65
View Mode: Grid [10 %]
Pothole Size Limit: 141.0
Object Count: 3
Max Object Width: 507.0
Object Confidence: 0.85
Inference: 444.4ms

KPI's

>60px

Width

>0.65

Confidence

<500ms

Processing Time

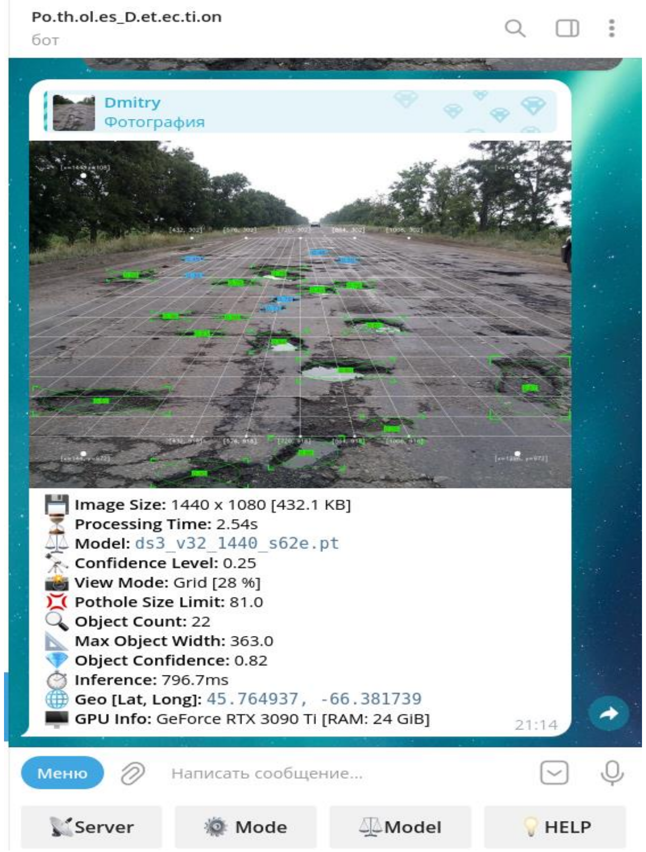


Image Size: 1440 x 1080 [432.1 KB]
Processing Time: 2.54s
Model: `ds3_v32_1440_s62e.pt`
Confidence Level: 0.25
View Mode: Grid [28 %]
Pothole Size Limit: 81.0
Object Count: 22
Max Object Width: 363.0
Object Confidence: 0.82
Inference: 796.7ms
Geo [Lat, Long]: `45.764937, -66.381739`
GPU Info: GeForce RTX 3090 Ti [RAM: 24 GiB]

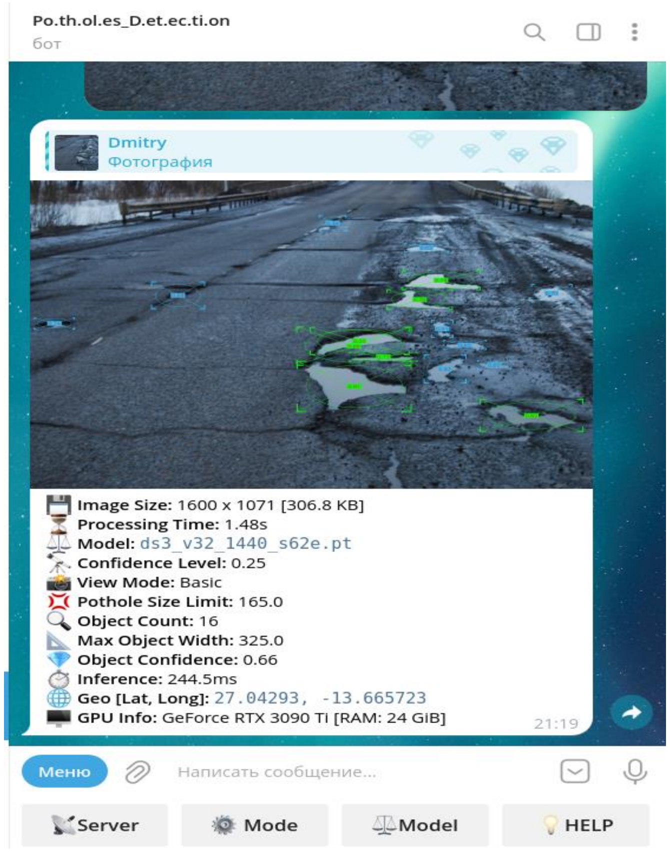


Image Size: 1600 x 1071 [306.8 KB]
Processing Time: 1.48s
Model: `ds3_v32_1440_s62e.pt`
Confidence Level: 0.25
View Mode: Basic
Pothole Size Limit: 165.0
Object Count: 16
Max Object Width: 325.0
Object Confidence: 0.66
Inference: 244.5ms
Geo [Lat, Long]: `27.04293, -13.665723`
GPU Info: GeForce RTX 3090 Ti [RAM: 24 GiB]

Achived Quality metrics for the YOLOv8

96,1%

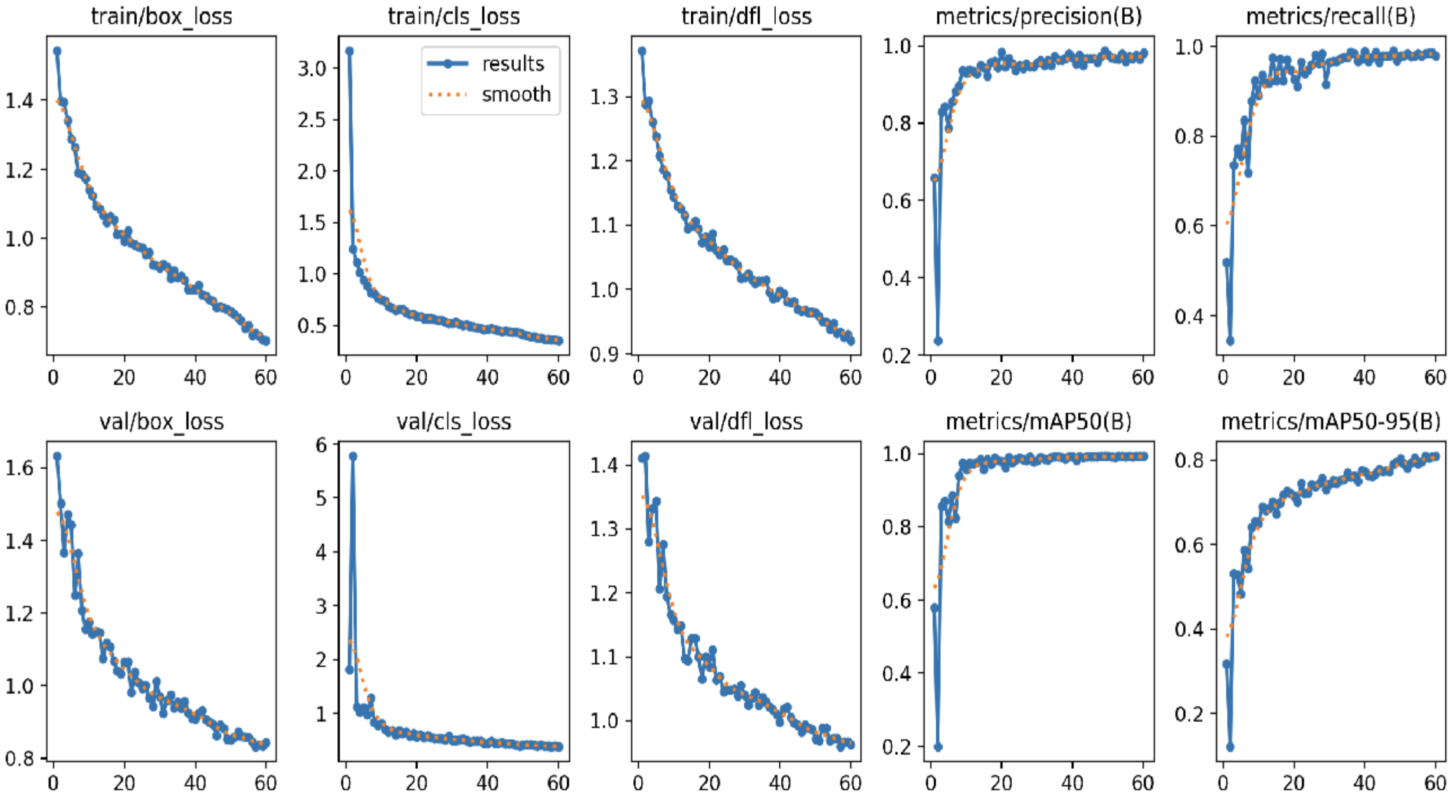
Precision

99,2%

Recall

97,6%

F1 Score



Plans

Future Plans and Developments

Enhancing Model Robustness

Scaling the System

Expanding Functionality

Conclusion

Conclusion

Successful Implementations

Effective Pothole Detection

Real-Time Processing and Feedback

User-Friendly Interface

Challenges and Unsuccessful Attempts

Handling Varied Lighting and Weather Conditions

Scalability Issues

Insights Gained

Importance of Data Quality

Importance of API in Progressive Development

User-Centric Design

Thanks!