EVA AI-Relational Database System

Georgia Tech Database Group

May 12, 2023

OVERVIEW

AI-Relational Database System | SQL meets Deep Learning

CHAPTER

ONE

WHAT IS EVA?

EVA is an **open-source AI-relational database with first-class support for deep learning models**. It aims to support AI-powered database applications that operate on both structured (tables) and unstructured data (videos, text, podcasts, PDFs, etc.) with deep learning models.

EVA accelerates AI pipelines using a collection of optimizations inspired by relational database systems including function caching, sampling, and cost-based operator reordering. It comes with a wide range of models for analyzing unstructured data including image classification, object detection, OCR, face detection, etc. It is fully implemented in Python, and licensed under the Apache license.

EVA supports a AI-oriented query language for analysing unstructured data. Here are some illustrative applications:

- Examining the emotion palette of actors in a movie
- Analysing traffic flow at an intersection
- Classifying images based on their content
- Recogizing license plates
- Analysing toxicity of social media memes

If you are wondering why you might need a video database system, start with page on Video Database Systems. It describes how EVA lets users easily make use of deep learning models and how they can reduce money spent on inference on large image or video datasets.

The Getting Started page shows how you can use EVA for different computer vision tasks, and how you can easily extend EVA to support your custom deep learning model in the form of user-defined functions.

The User Guides section contains Jupyter Notebooks that demonstrate how to use various features of EVA. Each notebook includes a link to Google Colab, where you can run the code by yourself.

CHAPTER

TWO

KEY FEATURES

1. With EVA, you can **easily combine SQL and deep learning models to build next-generation database applications**. EVA treats deep learning models as functions similar to traditional SQL functions like SUM().
2. EVA is **extensible by design**. You can write an **user-defined function** (UDF) that wraps arounds your custom deep learning model. In fact, all the built-in models that are included in EVA are written as user-defined functions.
3. EVA comes with a collection of **built-in sampling, caching, and filtering optimizations** inspired by relational database systems. These optimizations help **speed up queries on large datasets and save money spent on model inference**.

CHAPTER

THREE

NEXT STEPS

Getting Started A step-by-step guide to installing EVA and running queries

Query Language List of all the query commands supported by EVA

User Defined Functions A step-by-step tour of registering a user defined function that wraps around a custom deep learning model

CHAPTER

FOUR

ILLUSTRATIVE EVA APPLICATIONS

4.1 Traffic Analysis Application using Object Detection Model

4.2 MNIST Digit Recognition using Image Classification Model

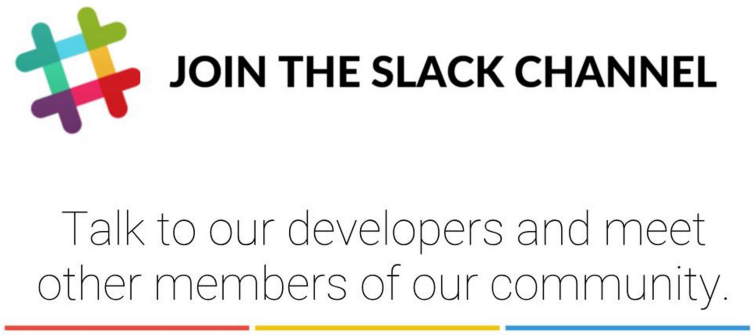
4.3 Movie Analysis Application using Face Detection + Emotion Classfication Models

CHAPTER

FIVE

COMMUNITY

Join the EVA community on Slack to ask questions and to share your ideas for improving EVA.



CHAPTER

SIX

GETTING STARTED

6.1 Part 1: Install EVA

EVA supports Python (versions >= 3.7). To install EVA, we recommend using the pip package manager:

```
pip install evadb
```

6.2 Launch EVA server

EVA is based on a client-server architecture. To launch the EVA server, run the following command on the terminal:

```
eva_server &
```

6.3 Part 2: Start a Jupyter Notebook Client

Here is an illustrative Jupyter notebook focusing on MNIST image classification using EVA. The notebook works on Google Colab.

6.3.1 Connect to the EVA server

To connect to the EVA server in the notebook, use the following Python code:

```
# allow nested asyncio calls for client to connect with server
import nest_asyncio
nest_asyncio.apply()
from eva.server.db_api import connect

# hostname and port of the server where EVA is running
connection = connect(host = '0.0.0.0', port = 5432)

# cursor allows the notebook client to send queries to the server
cursor = connection.cursor()
```

6.3.2 Load video for analysis

Download the MNIST video for analysis.

```
!wget -nc https://www.dropbox.com/s/yxljxz6zxoqu54v/mnist.mp4
```

Use the LOAD statement is used to load a video onto a table in EVA server.

```
cursor.execute('LOAD VIDEO "mnist.mp4" INTO MNISTVideoTable;')
response = cursor.fetch_all()
print(response)
```

6.4 Part 3: Register an user-defined function (UDF)

User-defined functions allow us to combine SQL with deep learning models. These functions wrap around deep learning models.

Download the user-defined function for classifying MNIST images.

```
!wget -nc https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/apps/
↪mnist/eva_mnist_udf.py
```

```
cursor.execute("""CREATE UDF IF NOT EXISTS MnistCNN
                  INPUT  (data NDARRAY (3, 28, 28))
                  OUTPUT (label TEXT(2))
                  TYPE  Classification
                  IMPL  'eva_mnist_udf.py';
                """)
response = cursor.fetch_all()
print(response)
```

6.4.1 Run a query using the newly registered UDF!

```
cursor.execute("""SELECT data, MnistCNN(data).label
                  FROM MNISTVideoTable
                  WHERE id = 30;""")
response = cursor.fetch_all()
```

6.4.2 Visualize the output

The output of the query is visualized in the notebook.

6.5 Part 5: Start a Command Line Client

Besides the notebook interface, EVA also exports a command line interface for querying the server. This interface allows for quick querying from the terminal:

```
>>> eva_client
eva=# LOAD VIDEO "mnist.mp4" INTO MNISTVid;
@status: ResponseStatus.SUCCESS
@batch:

0 Video successfully added at location: mnist.p4
@query_time: 0.045

eva=# SELECT id, data FROM MNISTVid WHERE id < 1000;
@status: ResponseStatus.SUCCESS
@batch:
            mnistvid.id     mnistvid.data
    0          0             [[[ 0 2 0]\n [0 0 0]\n...
    1          1             [[[ 2 2 0]\n [1 1 0]\n...
    2          2             [[[ 2 2 0]\n [1 2 2]\n...
    ..       ...
    997        997           [[[ 0 2 0]\n [0 0 0]\n...
    998        998           [[[ 0 2 0]\n [0 0 0]\n...
    999        999           [[[ 2 2 0]\n [1 1 0]\n...

[1000 rows x 2 columns]
@query_time: 0.216

eva=# exit
```

CHAPTER

SEVEN

LIST OF NOTEBOOKS

This section contains Jupyter Notebooks that demonstrate how to use various features of EVA. Each notebook includes a link to Google Colab, where you can run the code by yourself.

- Starting EVA Server
- MNIST Image Classification
- Object Detection with YOLO
- Emotion Analysis with PyTorch
- Image Segmentation with HuggingFace
- News Video Analysis with ChatGPT and Whisper

CHAPTER

EIGHT

START EVA SERVER

8.1 Launch EVA server

We use this notebook for launching the EVA server.

```
## Install EVA package if needed
%pip install "evadb" --quiet

import os
import time
from psutil import process_iter
from signal import SIGTERM
import re
import itertools

def shell(command):
    print(command)
    os.system(command)

def stop_eva_server():
    for proc in process_iter():
        if proc.name() == "eva_server":
            proc.send_signal(SIGTERM)

def is_eva_server_running():
    for proc in process_iter():
        if proc.name() == "eva_server":
            return True
    return False

def launch_eva_server():
    # Stop EVA server if it is running
    # stop_eva_server()

    os.environ['GPU_DEVICES'] = '0'

    # Start EVA server
    shell("nohup eva_server > eva.log 2>&1 &")

    last_few_lines_count = 3
```

(continues on next page)

(continued from previous page)

```
    try:
        with open('eva.log', 'r') as f:
            for lines in itertools.zip_longest(*[f]*last_few_lines_count):
                print(lines)
    except FileNotFoundError:
        pass

    # Wait for server to start
    time.sleep(10)

def connect_to_server():
    from eva.server.db_api import connect
    %pip install nest_asyncio --quiet
    import nest_asyncio
    nest_asyncio.apply()

    status = is_eva_server_running()
    if status == False:
        launch_eva_server()

    # Connect client with server
    connection = connect(host = '127.0.0.1', port = 5432)
    cursor = connection.cursor()

    return cursor

# Launch server
launch_eva_server()
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
nohup eva_server > eva.log 2>&1 &
```

CHAPTER

NINE

MNIST TUTORIAL

9.1 Start EVA server

We are reusing the start server notebook for launching the EVA server.

```
!wget -nc "https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/00-
↪start-eva-server.ipynb"
%run 00-start-eva-server.ipynb
cursor = connect_to_server()
```

```
File '00-start-eva-server.ipynb' already there; not retrieving.
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
nohup eva_server > eva.log 2>&1 &
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

9.2 Downloading the videos

```
# Getting MNIST as a video
!wget -nc https://www.dropbox.com/s/yxljxz6zxoqu54v/mnist.mp4
# Getting a udf
!wget -nc https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/apps/
↪mnist/eva_mnist_udf.py
```

```
File 'mnist.mp4' already there; not retrieving.
```

```
File 'eva_mnist_udf.py' already there; not retrieving.
```

9.3 Upload the video for analysis

```
cursor.execute('DROP TABLE IF EXISTS MNISTVid')
response = cursor.fetch_all()
response.as_df()
cursor.execute("LOAD VIDEO 'mnist.mp4' INTO MNISTVid")
response = cursor.fetch_all()
response.as_df()
```

```
                            0
0  Number of loaded VIDEO: 1
```

9.4 Visualize Video

```
from IPython.display import Video
Video("mnist.mp4", embed=True)
```

```
<IPython.core.display.Video object>
```

9.5 Create an user-defined function (UDF) for analyzing the frames

```
cursor.execute("""CREATE UDF IF NOT EXISTS
                  MnistCNN
                  INPUT  (data NDARRAY (3, 28, 28))
                  OUTPUT (label TEXT(2))
                  TYPE  Classification
                  IMPL  'eva_mnist_udf.py'
        """)
response = cursor.fetch_all()
response.as_df()
```

```
                                                   0
0  UDF MnistCNN successfully added to the database.
```

9.6 Run the Image Classification UDF on video

```
cursor.execute("""SELECT data, MnistCNN(data).label
                  FROM MNISTVid
                  WHERE id = 30 OR id = 50 OR id = 70 OR id = 0 OR id = 140""")
response = cursor.fetch_all()
response.as_df()
```

```
                                      mnistvid.data mnistcnn.label
0  [[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...              6
```

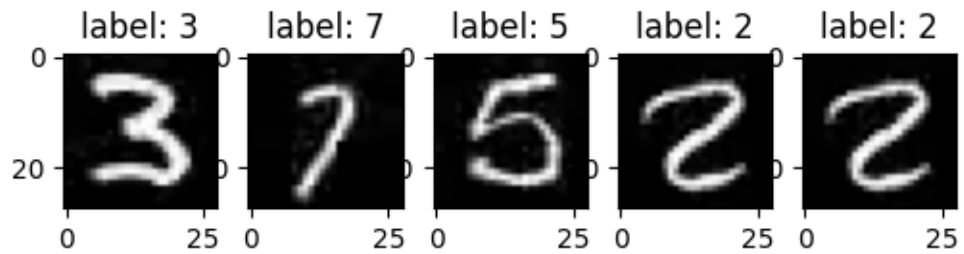
(continues on next page)

(continued from previous page)

```
1  [[[2, 2, 2], [2, 2, 2], [2, 2, 2], [2, 2, 2], ...              2
2  [[[13, 13, 13], [2, 2, 2], [2, 2, 2], [13, 13,...              3
3  [[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...              7
4  [[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ...              5
```

9.7 Visualize output of query on the video

```
# !pip install matplotlib
import matplotlib.pyplot as plt
import numpy as np

# create figure (fig), and array of axes (ax)
fig, ax = plt.subplots(nrows=1, ncols=5, figsize=[6,8])

df = response.batch.frames
for axi in ax.flat:
    idx = np.random.randint(len(df))
    img = df['mnistvid.data'].iloc[idx]
    label = df['mnistcnn.label'].iloc[idx]
    axi.imshow(img)

    axi.set_title(f'label: {label}')

plt.show()
```



CHAPTER

TEN

OBJECT DETECTION TUTORIAL

10.1 Start EVA server

We are reusing the start server notebook for launching the EVA server.

```
!wget -nc "https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/00-
↪start-eva-server.ipynb"
%run 00-start-eva-server.ipynb
cursor = connect_to_server()
```

```
File '00-start-eva-server.ipynb' already there; not retrieving.
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
nohup eva_server > eva.log 2>&1 &
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

10.2 Download the Videos

```
# Getting the video files
!wget -nc "https://www.dropbox.com/s/k00wge9exwkfxz6/ua_detrac.mp4?raw=1" -O ua_detrac.
↪mp4
```

```
File 'ua_detrac.mp4' already there; not retrieving.
```

10.3 Load the surveillance videos for analysis

10.3.1 We use regular expression to load all the videos into the table

```
cursor.execute('DROP TABLE IF EXISTS ObjectDetectionVideos')
response = cursor.fetch_all()
response.as_df()

cursor.execute('LOAD VIDEO "ua_detrac.mp4" INTO ObjectDetectionVideos;')
response = cursor.fetch_all()
response.as_df()
```

```
                           0
0  Number of loaded VIDEO: 1
```

10.4 Visualize Video

```
from IPython.display import Video
Video("ua_detrac.mp4", embed=True)
```

```
<IPython.core.display.Video object>
```

10.5 Register YOLO Object Detector an an User-Defined Function (UDF) in EVA

```
cursor.execute("""
            CREATE UDF IF NOT EXISTS Yolo
            TYPE  ultralytics
            'model' 'yolov8m.pt';
      """)
response = cursor.fetch_all()
response.as_df()
```

```
                                        0
0  UDF Yolo already exists, nothing added.
```

10.6 Run Object Detector on the video

```
cursor.execute("""SELECT id, Yolo(data)
                  FROM ObjectDetectionVideos
                  WHERE id < 20""")
response = cursor.fetch_all()
response.as_df()
```

```
    objectdetectionvideos.id  \
0                          0
1                          1
2                          2
3                          3
4                          4
5                          5
6                          6
7                          7
8                          8
9                          9
10                        10
11                        11
12                        12
13                        13
14                        14
15                        15
16                        16
17                        17
18                        18
19                        19

                                        yolo.labels  \
0   [car, car, car, car, car, car, person, car, ca...
1   [car, car, car, car, car, car, car, car, car, ...
2   [car, car, car, car, car, car, car, person, ca...
3   [car, car, car, car, car, car, car, car, car, ...
4   [car, car, car, car, car, car, car, car, car, ...
5   [car, car, car, car, car, car, person, car, ca...
6   [car, car, car, car, car, car, car, person, ca...
7   [car, car, car, car, car, car, car, car, car, ...
8   [car, car, car, car, car, car, person, car, ca...
9   [car, car, car, car, car, car, car, person, ca...
10  [car, car, car, car, car, car, car, person, ca...
11  [car, car, car, car, car, car, person, car, ca...
12  [car, car, car, car, car, car, car, person, ca...
13  [car, car, car, car, car, car, person, car, ca...
14  [car, car, car, car, car, car, person, car, ca...
15  [car, car, car, car, car, car, person, car, ca...
16  [car, car, car, car, car, car, car, person, ca...
17  [car, car, car, car, car, car, car, person, ca...
18  [car, car, car, car, car, car, car, person, mo...
19  [car, car, car, car, car, person, car, car, ca...
```

(continues on next page)

(continued from previous page)

```
                                              yolo.bboxes  \
0   [[829.0, 277.0, 960.0, 360.0], [615.0, 216.0, ...
1   [[832.0, 278.0, 960.0, 361.0], [616.0, 216.0, ...
2   [[836.0, 279.0, 960.0, 362.0], [618.0, 216.0, ...
3   [[839.0, 280.0, 960.0, 363.0], [619.0, 217.0, ...
4   [[843.0, 281.0, 960.0, 364.0], [621.0, 218.0, ...
5   [[847.0, 282.0, 960.0, 363.0], [623.0, 218.0, ...
6   [[851.0, 283.0, 959.0, 360.0], [625.0, 219.0, ...
7   [[855.0, 284.0, 960.0, 357.0], [626.0, 220.0, ...
8   [[859.0, 285.0, 960.0, 357.0], [628.0, 221.0, ...
9   [[863.0, 286.0, 960.0, 357.0], [630.0, 222.0, ...
10  [[632.0, 223.0, 744.0, 284.0], [867.0, 287.0, ...
11  [[871.0, 289.0, 960.0, 356.0], [634.0, 223.0, ...
12  [[636.0, 223.0, 750.0, 287.0], [875.0, 290.0, ...
13  [[171.0, 409.0, 291.0, 539.0], [637.0, 224.0, ...
14  [[174.0, 405.0, 294.0, 538.0], [885.0, 291.0, ...
15  [[888.0, 293.0, 960.0, 355.0], [177.0, 400.0, ...
16  [[893.0, 293.0, 960.0, 355.0], [180.0, 396.0, ...
17  [[182.0, 392.0, 296.0, 519.0], [897.0, 294.0, ...
18  [[901.0, 295.0, 960.0, 356.0], [647.0, 225.0, ...
19  [[648.0, 226.0, 770.0, 293.0], [906.0, 297.0, ...

                                              yolo.scores
0   [0.91, 0.86, 0.85, 0.83, 0.76, 0.73, 0.72, 0.7...
1   [0.92, 0.85, 0.84, 0.83, 0.78, 0.76, 0.76, 0.7...
2   [0.92, 0.84, 0.84, 0.82, 0.81, 0.75, 0.73, 0.7...
3   [0.91, 0.84, 0.82, 0.8, 0.8, 0.75, 0.74, 0.72,...
4   [0.9, 0.85, 0.83, 0.8, 0.76, 0.73, 0.72, 0.72,...
5   [0.89, 0.86, 0.84, 0.8, 0.78, 0.74, 0.72, 0.72...
6   [0.89, 0.87, 0.85, 0.81, 0.79, 0.73, 0.72, 0.7...
7   [0.9, 0.87, 0.84, 0.83, 0.83, 0.79, 0.73, 0.67...
8   [0.89, 0.88, 0.83, 0.82, 0.79, 0.71, 0.68, 0.6...
9   [0.88, 0.87, 0.84, 0.82, 0.8, 0.75, 0.74, 0.74...
10  [0.88, 0.88, 0.85, 0.82, 0.8, 0.79, 0.76, 0.71...
11  [0.9, 0.9, 0.85, 0.8, 0.79, 0.77, 0.69, 0.68, ...
12  [0.9, 0.88, 0.83, 0.81, 0.78, 0.78, 0.78, 0.67...
13  [0.9, 0.89, 0.89, 0.83, 0.81, 0.81, 0.72, 0.71...
14  [0.9, 0.89, 0.88, 0.84, 0.82, 0.81, 0.75, 0.72...
15  [0.89, 0.88, 0.87, 0.84, 0.82, 0.78, 0.76, 0.7...
16  [0.88, 0.88, 0.87, 0.82, 0.81, 0.76, 0.75, 0.7...
17  [0.9, 0.89, 0.87, 0.83, 0.82, 0.78, 0.72, 0.69...
18  [0.88, 0.88, 0.83, 0.82, 0.8, 0.78, 0.75, 0.7,...
19  [0.89, 0.87, 0.81, 0.8, 0.78, 0.77, 0.73, 0.72...
```

10.7 Visualizing output of the Object Detector on the video

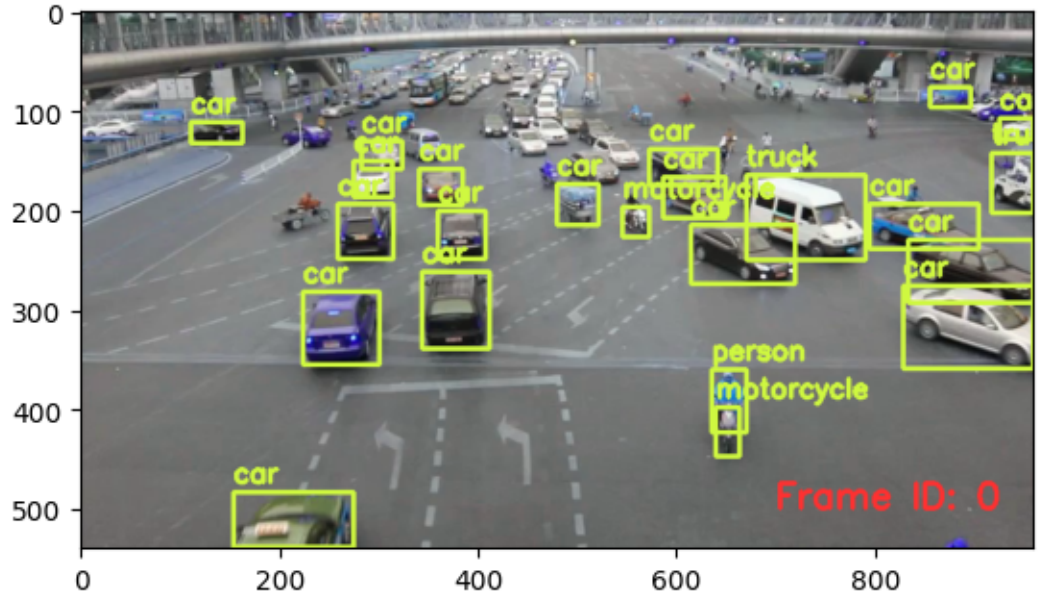
```
import cv2
from pprint import pprint
from matplotlib import pyplot as plt

def annotate_video(detections, input_video_path, output_video_path):
    color1=(207, 248, 64)
    color2=(255, 49, 49)
    thickness=4

    vcap = cv2.VideoCapture(input_video_path)
    width = int(vcap.get(3))
    height = int(vcap.get(4))
    fps = vcap.get(5)
    fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v') #codec
    video=cv2.VideoWriter(output_video_path, fourcc, fps, (width,height))

    frame_id = 0
    # Capture frame-by-frame
    # ret = 1 if the video is captured; frame is the image
    ret, frame = vcap.read()

    while ret:
        df = detections
        df = df[['yolo.bboxes', 'yolo.labels']][df.index == frame_id]
        if df.size:
            dfLst = df.values.tolist()
            for bbox, label in zip(dfLst[0][0], dfLst[0][1]):
                x1, y1, x2, y2 = bbox
                x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
                # object bbox
                frame=cv2.rectangle(frame, (x1, y1), (x2, y2), color1, thickness)
                # object label
                cv2.putText(frame, label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,␣
↪color1, thickness)
                # frame label
                cv2.putText(frame, 'Frame ID: ' + str(frame_id), (700, 500), cv2.FONT_
↪HERSHEY_SIMPLEX, 1.2, color2, thickness)
            video.write(frame)

            # Stop after twenty frames (id < 20 in previous query)
            if frame_id == 20:
                break

            # Show every fifth frame
            if frame_id % 5 == 0:
                plt.imshow(frame)
                plt.show()


        frame_id+=1
```

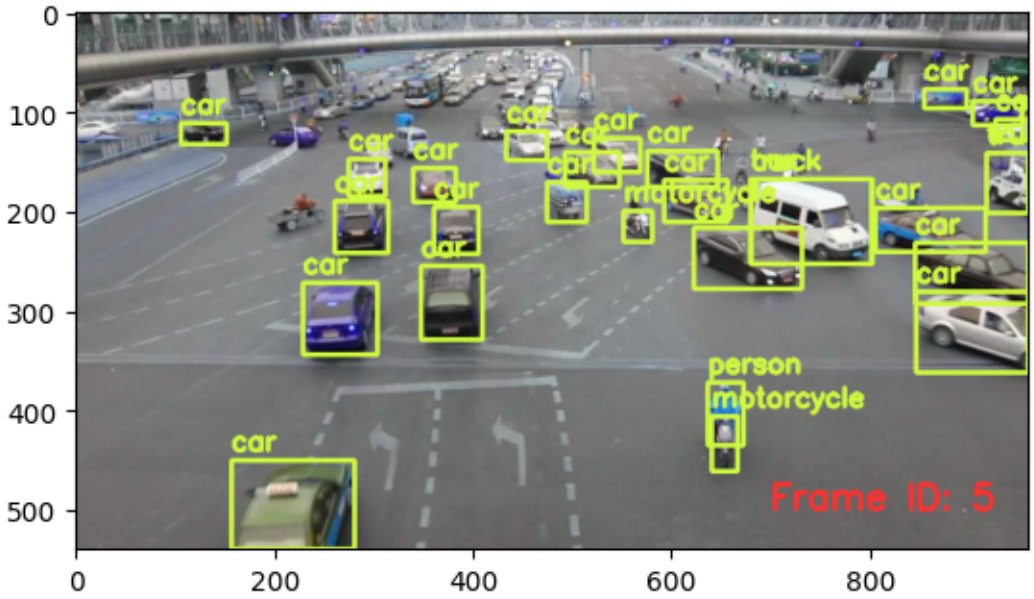
(continues on next page)

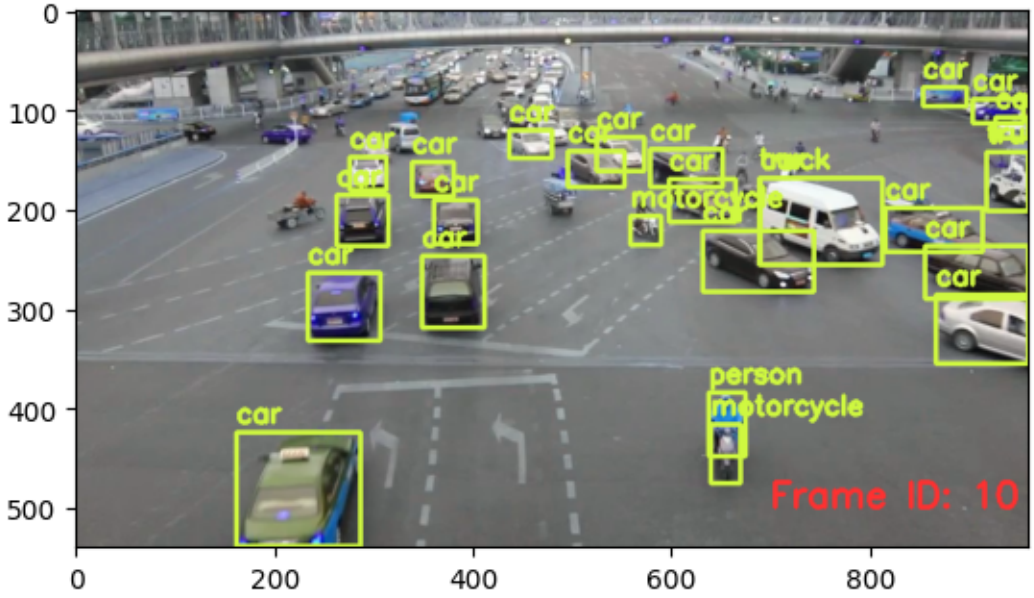
(continued from previous page)

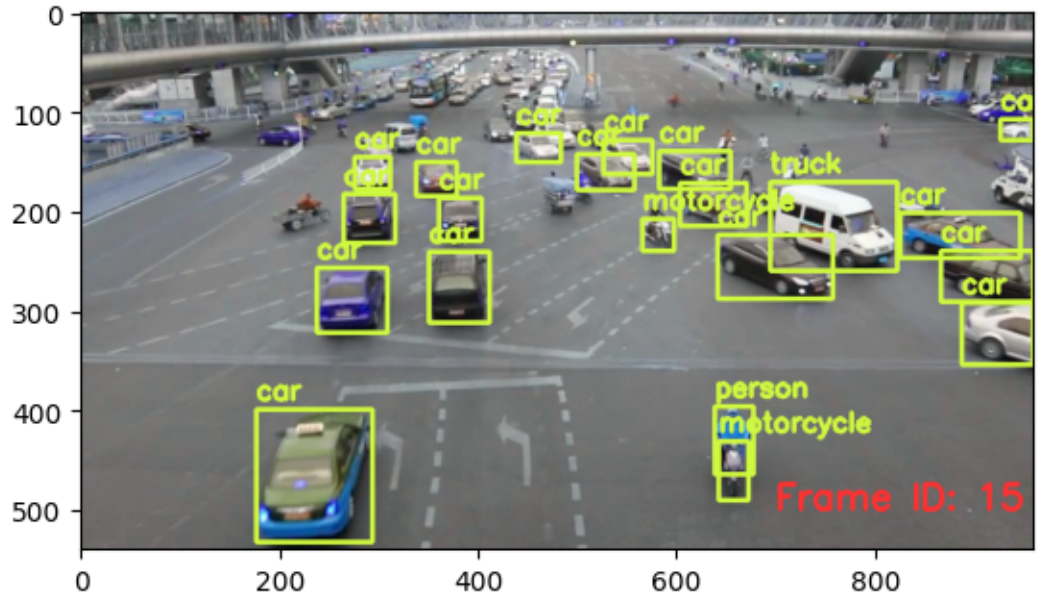
```
        ret, frame = vcap.read()

    video.release()
    vcap.release()
```

```
from ipywidgets import Video, Image
input_path = 'ua_detrac.mp4'
output_path = 'video.mp4'

dataframe = response.as_df()
annotate_video(dataframe, input_path, output_path)
Video.from_file(output_path)
```




0
100
200
300
400
500
0
200
400
600
800
car
person
motorcycle
Frame ID: 5


0
100
200
300
400
500
0
200
400
600
800
car
person
motorcycle
Frame ID: 10




```
Video(value=b'\x00\x00\x00\x1cftypisom\x00\x00\x02\x00isomiso2mp41\x00\x00\x00\x08free\
↪x00\tI\x95mdat\x00\x00\...
```

10.8 Dropping an User-Defined Function (UDF)

```
cursor.execute("DROP UDF IF EXISTS Yolo;")
response = cursor.fetch_all()
response.as_df()
```

```
                                0
0  UDF Yolo successfully dropped
```

CHAPTER

ELEVEN

EMOTION ANALYSIS

11.1 Start EVA Server

We are reusing the start server notebook for launching the EVA server

```
!wget -nc "https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/00-
↪start-eva-server.ipynb"
%run 00-start-eva-server.ipynb
cursor = connect_to_server()
```

```
File '00-start-eva-server.ipynb' already there; not retrieving.
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
nohup eva_server > eva.log 2>&1 &
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

11.2 Video Files

getting some video files to test

```
# A video of a happy person
!wget -nc "https://www.dropbox.com/s/gzfhwmib7u804zy/defhappy.mp4?raw=1" -O defhappy.mp4

# Adding Emotion detection
!wget -nc https://raw.githubusercontent.com/georgia-tech-db/eva/master/eva/udfs/emotion_
↪detector.py

# Adding Face Detector
```

(continues on next page)

(continued from previous page)

```
!wget -nc https://raw.githubusercontent.com/georgia-tech-db/eva/master/eva/udfs/face_
↪detector.py
```

```
File 'defhappy.mp4' already there; not retrieving.
```

```
File 'emotion_detector.py' already there; not retrieving.
```

```
File 'face_detector.py' already there; not retrieving.
```

11.3 Adding the video file to EVADB for analysis

```
cursor.execute('DROP TABLE IF EXISTS HAPPY')
response = cursor.fetch_all()
response.as_df()
cursor.execute('LOAD VIDEO "defhappy.mp4" INTO HAPPY')
response = cursor.fetch_all()
response.as_df()
```

```
                          0
0  Number of loaded VIDEO: 1
```

11.4 Visualize Video

```
from IPython.display import Video
Video("defhappy.mp4", height=450, width=800, embed=True)
```

```
<IPython.core.display.Video object>
```

11.5 Create an user-defined function(UDF) for analyzing the frames

```
cursor.execute("""CREATE UDF IF NOT EXISTS EmotionDetector
        INPUT (frame NDARRAY UINT8(3, ANYDIM, ANYDIM))
        OUTPUT (labels NDARRAY STR(ANYDIM), scores NDARRAY FLOAT32(ANYDIM))
        TYPE  Classification IMPL 'emotion_detector.py';
        """)
response = cursor.fetch_all()
response.as_df()

cursor.execute("""CREATE UDF IF NOT EXISTS FaceDetector
                  INPUT  (frame NDARRAY UINT8(3, ANYDIM, ANYDIM))
                  OUTPUT (bboxes NDARRAY FLOAT32(ANYDIM, 4),
                          scores NDARRAY FLOAT32(ANYDIM))
                  TYPE  FaceDetection
```

(continues on next page)

(continued from previous page)

```
                IMPL  'face_detector.py';
        """)
response = cursor.fetch_all()
response.as_df()
```

```
                                                   0
0  UDF FaceDetector successfully added to the dat...
```

11.6 Run the Face Detection UDF on video

```
cursor.execute("""SELECT id, FaceDetector(data)
                  FROM HAPPY WHERE id<10""")
response = cursor.fetch_all()
response.as_df()
```

```
   happy.id                          facedetector.bboxes  \
0         0  [[502, 94, 762, 435], [238, 296, 325, 398]]
1         1                         [[501, 96, 763, 435]]
2         2                         [[504, 97, 766, 437]]
3         3                         [[498, 90, 776, 446]]
4         4                         [[496, 99, 767, 444]]
5         5  [[499, 87, 777, 448], [236, 305, 324, 407]]
6         6                         [[500, 89, 778, 449]]
7         7                         [[501, 89, 781, 452]]
8         8                         [[503, 90, 783, 450]]
9         9                         [[508, 87, 786, 447]]

        facedetector.scores
0  [0.99990165, 0.79820246]
1                [0.999918]
2               [0.9999138]
3              [0.99996686]
4               [0.9999982]
5    [0.9999136, 0.8369736]
6               [0.9999131]
7               [0.9999124]
8              [0.99994683]
9                [0.999949]
```

11.7 Run the Emotion Detection UDF on the outputs of the Face Detection UDF

```
cursor.execute("""SELECT id, bbox, EmotionDetector(Crop(data, bbox))
                  FROM HAPPY JOIN LATERAL  UNNEST(FaceDetector(data)) AS Face(bbox,␣
↪conf)
                  WHERE id < 15;""")
response = cursor.fetch_all()
response.as_df()
```

```
    happy.id             Face.bbox emotiondetector.labels  \
0          0   [502, 94, 762, 435]                   happy
1          0  [238, 296, 325, 398]                 neutral
2          1   [501, 96, 763, 435]                   happy
3          2   [504, 97, 766, 437]                   happy
4          3   [498, 90, 776, 446]                   happy
5          4   [496, 99, 767, 444]                   happy
6          5   [499, 87, 777, 448]                   happy
7          5  [236, 305, 324, 407]                 neutral
8          6   [500, 89, 778, 449]                   happy
9          7   [501, 89, 781, 452]                   happy
10         8   [503, 90, 783, 450]                   happy
11         9   [508, 87, 786, 447]                   happy
12        10   [505, 86, 788, 452]                   happy
13        10  [235, 309, 322, 411]                 neutral
14        11   [514, 85, 790, 454]                   happy
15        12   [514, 86, 790, 454]                   happy
16        13   [515, 87, 790, 454]                   happy
17        14   [516, 86, 792, 455]                   happy

    emotiondetector.scores
0                 0.999642
1                 0.780949
2                 0.999644
3                 0.999668
4                 0.999654
5                 0.999649
6                 0.999710
7                 0.760779
8                 0.999671
9                 0.999671
10                0.999689
11                0.999691
12                0.999729
13                0.407872
14                0.999745
15                0.999729
16                0.999718
17                0.999739
```

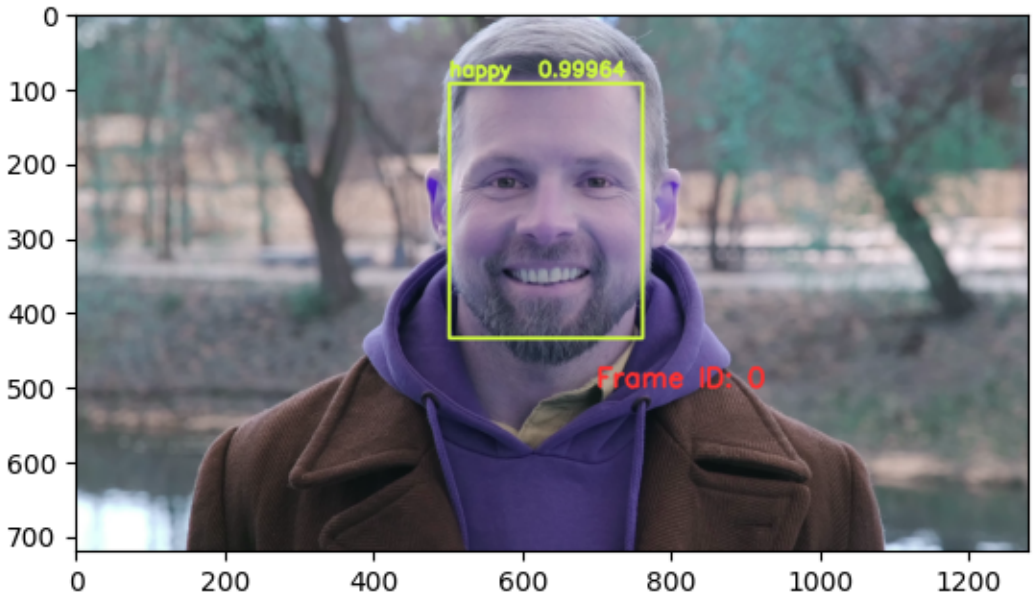
```
import cv2
from pprint import pprint
from matplotlib import pyplot as plt

def annotate_video(detections, input_video_path, output_video_path):
    color1=(207, 248, 64)
    color2=(255, 49, 49)
    thickness=4

    vcap = cv2.VideoCapture(input_video_path)
    width = int(vcap.get(3))
    height = int(vcap.get(4))
    fps = vcap.get(5)
    fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v') #codec
    video=cv2.VideoWriter(output_video_path, fourcc, fps, (width,height))

    frame_id = 0
    # Capture frame-by-frame
    # ret = 1 if the video is captured; frame is the image
    ret, frame = vcap.read()

    while ret:
        df = detections
        df = df[['Face.bbox', 'emotiondetector.labels', 'emotiondetector.scores']][df.
↪index == frame_id]
        if df.size:

            x1, y1, x2, y2 = df['Face.bbox'].values[0]
            label = df['emotiondetector.labels'].values[0]
            score = df['emotiondetector.scores'].values[0]
            x1, y1, x2, y2 = int(x1), int(y1), int(x2), int(y2)
            # object bbox
            frame=cv2.rectangle(frame, (x1, y1), (x2, y2), color1, thickness)
            # object label
            cv2.putText(frame, label, (x1, y1-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, color1,
↪ thickness)
            # object score
            cv2.putText(frame, str(round(score, 5)), (x1+120, y1-10), cv2.FONT_HERSHEY_
↪SIMPLEX, 0.9, color1, thickness)
            # frame label
            cv2.putText(frame, 'Frame ID: ' + str(frame_id), (700, 500), cv2.FONT_
↪HERSHEY_SIMPLEX, 1.2, color2, thickness)

            video.write(frame)
            # Show every fifth frame
            if frame_id % 5 == 0:
                plt.imshow(frame)
                plt.show()

        frame_id+=1
        ret, frame = vcap.read()

    video.release()
```

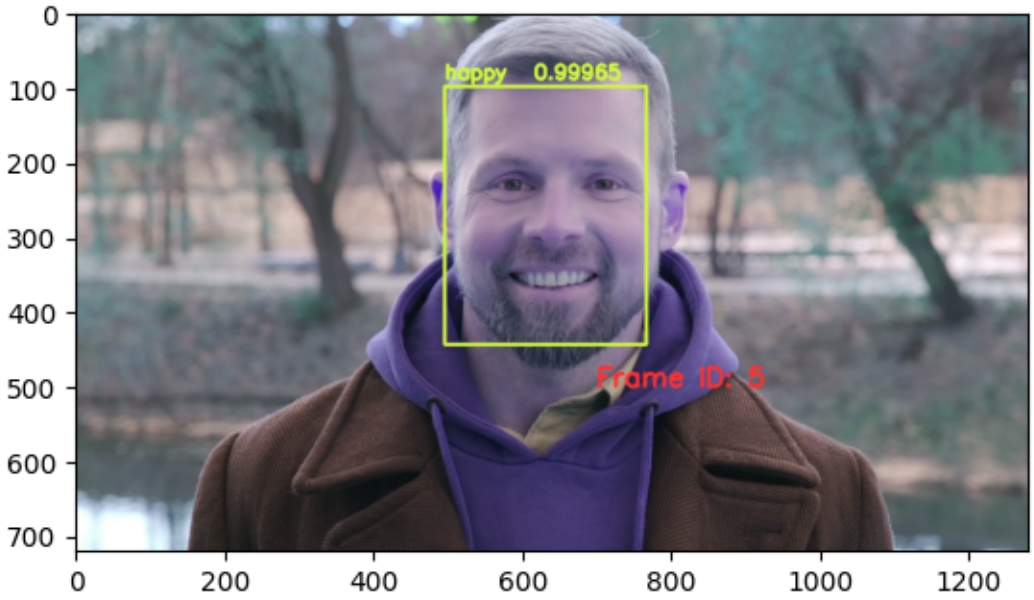
(continues on next page)

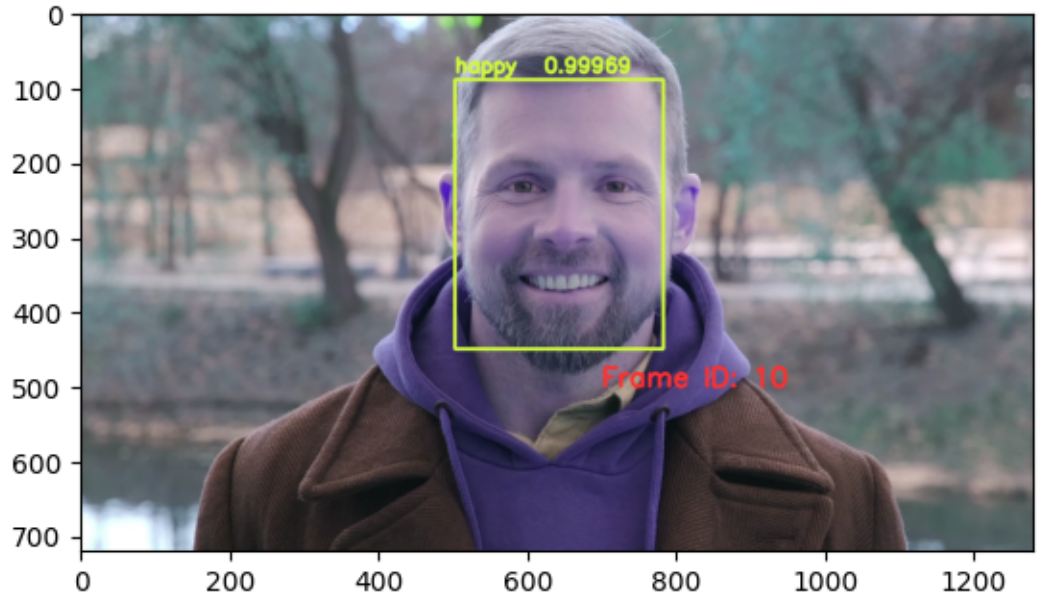
(continued from previous page)

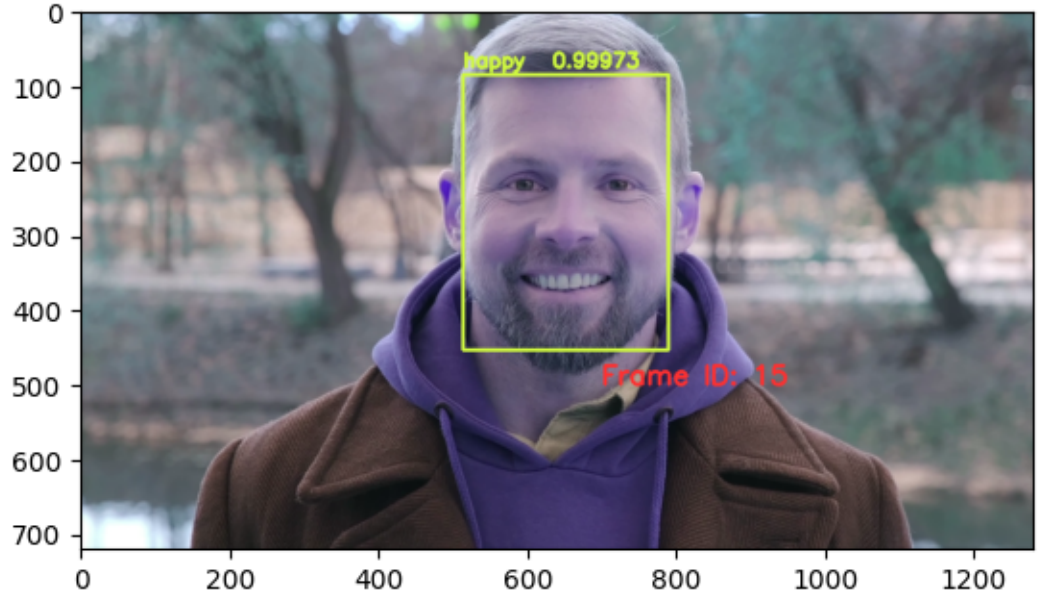
```
    vcap.release()
```

```
from ipywidgets import Video, Image
input_path = 'defhappy.mp4'
output_path = 'video.mp4'

dataframe = response.as_df()
annotate_video(dataframe, input_path, output_path)
```






happy 0.99969
Frame ID: 10
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200


happy 0.99973
Frame ID: 15
0
100
200
300
400
500
600
700
0
200
400
600
800
1000
1200

CHAPTER

TWELVE

IMAGE SEGMENTATION TUTORIAL

12.1 Start EVA server

We are reusing the start server notebook for launching the EVA server.

```
!wget -nc "https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/00-
↪start-eva-server.ipynb"
!pip install timm
%run 00-start-eva-server.ipynb
stop_eva_server()
cursor = connect_to_server()
```

```
File '00-start-eva-server.ipynb' already there; not retrieving.
```

```
Requirement already satisfied: timm in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (0.6.13)
Requirement already satisfied: huggingface-hub in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from timm) (0.14.1)
Requirement already satisfied: pyyaml in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from timm) (6.0)
Requirement already satisfied: torchvision in /home/jarulraj3/eva/test_evadb/lib/python3.
↪10/site-packages (from timm) (0.14.0)
Requirement already satisfied: torch>=1.7 in /home/jarulraj3/eva/test_evadb/lib/python3.
↪10/site-packages (from timm) (1.13.0)
Requirement already satisfied: nvidia-cuda-runtime-cu11==11.7.99 in /home/jarulraj3/eva/
↪test_evadb/lib/python3.10/site-packages (from torch>=1.7->timm) (11.7.99)
Requirement already satisfied: typing-extensions in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from torch>=1.7->timm) (4.4.0)
Requirement already satisfied: nvidia-cuda-nvrtc-cu11==11.7.99 in /home/jarulraj3/eva/
↪test_evadb/lib/python3.10/site-packages (from torch>=1.7->timm) (11.7.99)
Requirement already satisfied: nvidia-cublas-cu11==11.10.3.66 in /home/jarulraj3/eva/
↪test_evadb/lib/python3.10/site-packages (from torch>=1.7->timm) (11.10.3.66)
Requirement already satisfied: nvidia-cudnn-cu11==8.5.0.96 in /home/jarulraj3/eva/test_
↪evadb/lib/python3.10/site-packages (from torch>=1.7->timm) (8.5.0.96)
Requirement already satisfied: wheel in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from nvidia-cublas-cu11==11.10.3.66->torch>=1.7->timm) (0.38.4)
Requirement already satisfied: setuptools in /home/jarulraj3/eva/test_evadb/lib/python3.
↪10/site-packages (from nvidia-cublas-cu11==11.10.3.66->torch>=1.7->timm) (65.6.0)
```

```
Requirement already satisfied: packaging>=20.9 in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from huggingface-hub->timm) (23.0)
Requirement already satisfied: requests in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from huggingface-hub->timm) (2.28.1)
Requirement already satisfied: fsspec in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from huggingface-hub->timm) (2023.4.0)
Requirement already satisfied: filelock in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from huggingface-hub->timm) (3.8.0)
Requirement already satisfied: tqdm>=4.42.1 in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from huggingface-hub->timm) (4.64.1)
```

```
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /home/jarulraj3/eva/test_evadb/
↪lib/python3.10/site-packages (from torchvision->timm) (9.0.1)
Requirement already satisfied: numpy in /home/jarulraj3/eva/test_evadb/lib/python3.10/
↪site-packages (from torchvision->timm) (1.23.4)
```

```
Requirement already satisfied: certifi>=2017.4.17 in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from requests->huggingface-hub->timm) (2022.9.24)
Requirement already satisfied: charset-normalizer<3,>=2 in /home/jarulraj3/eva/test_
↪evadb/lib/python3.10/site-packages (from requests->huggingface-hub->timm) (2.1.1)
Requirement already satisfied: idna<4,>=2.5 in /home/jarulraj3/eva/test_evadb/lib/
↪python3.10/site-packages (from requests->huggingface-hub->timm) (3.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /home/jarulraj3/eva/test_evadb/
↪lib/python3.10/site-packages (from requests->huggingface-hub->timm) (1.26.12)
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
nohup eva_server > eva.log 2>&1 &
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

12.2 Download the Videos

```
# # Getting the video files
!wget -nc "https://www.dropbox.com/s/k00wge9exwkfxz6/ua_detrac.mp4?raw=1" -O ua_detrac.
↪mp4
```

```
File ‘ua_detrac.mp4’ already there; not retrieving.
```

12.3 Load sample video from DAVIS dataset for analysis

```
cursor.execute('DROP TABLE IF EXISTS VideoForSegmentation;')
response = cursor.fetch_all()
response.as_df()
cursor.execute('LOAD VIDEO "ua_detrac.mp4" INTO VideoForSegmentation')
response = cursor.fetch_all()
response.as_df()
```

```
                            0
0  Number of loaded VIDEO: 1
```

12.4 Visualize Video

```
from IPython.display import Video
Video("ua_detrac.mp4", embed=True)
```

```
<IPython.core.display.Video object>
```

12.5 Register Hugging Face Segmentation Model as an User-Defined Function (UDF) in EVA

```
### Using HuggingFace with EVA requires specifying the task
### The task here is 'image-segmentation'
### The model is 'facebook/detr-resnet-50-panoptic'
cursor.execute("""CREATE UDF IF NOT EXISTS HFSegmentation
      TYPE HuggingFace
      'task' 'image-segmentation'
      'model' 'facebook/detr-resnet-50-panoptic'
      """)
response = cursor.fetch_all()
response.as_df()
```

```
                                                   0
0  UDF HFSegmentation successfully added to the d...
```

12.6 Run Image Segmentation on the video

```
cursor.execute("""SELECT HFSegmentation(data)
                  FROM VideoForSegmentation SAMPLE 5
                  WHERE id < 20""")
response = cursor.fetch_all()
response.as_df()
```

```
                                hfsegmentation.score  \
0  [0.906596, 0.989519, 0.960914, 0.923789, 0.960...
1  [0.985118, 0.963139, 0.963819, 0.960939, 0.926...
2  [0.989573, 0.900049, 0.966254, 0.96056, 0.9388...
3  [0.913261, 0.949733, 0.943763, 0.98639, 0.9744...

                                hfsegmentation.label  \
0  [motorcycle, motorcycle, person, car, car, per...
1  [motorcycle, person, car, car, person, bridge,...
2  [motorcycle, person, person, car, car, car, pe...
3  [truck, person, car, car, car, car, car, perso...

                                 hfsegmentation.mask
0  [<PIL.Image.Image image mode=L size=960x540 at...
1  [<PIL.Image.Image image mode=L size=960x540 at...
2  [<PIL.Image.Image image mode=L size=960x540 at...
3  [<PIL.Image.Image image mode=L size=960x540 at...
```

12.7 Visualizing output of the Image Segmenter on the video

```
import numpy as np
from PIL import Image
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt
import cv2

def get_color_mapping(all_labels):
    unique_labels = set(label for labels in all_labels for label in labels)
    num_colors = len(unique_labels)
    colormap = plt.colormaps["tab20"]
    colors = [colormap(i % 20)[:3] for i in range(num_colors)]
    colors = [tuple(int(c * 255) for c in color) for color in colors]
    color_mapping = {label: color for label, color in zip(unique_labels, colors)}
    return  color_mapping

def annotate_single_frame(frame, segments, labels, color_mapping):
    overlay = np.zeros_like(frame)

    # Overlay segments
    for mask, label in zip(segments, labels):
        mask_np = np.array(mask).astype(bool)
        overlay[mask_np] = color_mapping[label]

    # Combine original frame with overlay
    new_frame = Image.blend(
        Image.fromarray(frame.astype(np.uint8)),
        Image.fromarray(overlay.astype(np.uint8)),
        alpha=0.5,
    )
```

(continues on next page)

(continued from previous page)

```
    return new_frame

def annotate_video(segmentations, input_video_path, output_video_path, model_name =
↪'hfsegmentation'):
    all_segments = segmentations[f'{model_name}.mask']
    all_labels = segmentations[f'{model_name}.label']


    color_mapping = get_color_mapping(all_labels)

    vcap = cv2.VideoCapture(input_video_path)
    width = int(vcap.get(3))
    height = int(vcap.get(4))
    fps = vcap.get(5)
    fourcc = cv2.VideoWriter_fourcc('m', 'p', '4', 'v') #codec
    video=cv2.VideoWriter(output_video_path, fourcc, fps, (width,height))

    frame_id = 0
    ret, frame = vcap.read()
    while ret and frame_id < len(all_segments):
        segments = all_segments[frame_id]
        labels = all_labels[frame_id]
        new_frame = annotate_single_frame(frame, segments, labels, color_mapping)
        video.write(np.array(new_frame))
        if frame_id % 5 == 0:
            legend_patches = [mpatches.Patch(color=np.array(color_mapping[label])/255,␣
↪label=label) for label in set(labels)]
            plt.imshow(new_frame)
            plt.legend(handles=legend_patches, bbox_to_anchor=(1.05, 1), loc='upper left
↪', borderaxespad=0.)
            plt.axis('off')
            plt.tight_layout()
            plt.show()


        frame_id += 1
        ret, frame = vcap.read()

    video.release()
    vcap.release()
```

```
from ipywidgets import Video
input_path = 'ua_detrac.mp4'
output_path = 'video.mp4'

dataframe = response.as_df()
annotate_video(dataframe, input_path, output_path)
Video.from_file(output_path)
```


```
Video(value=b'\x00\x00\x00\x1cftypisom\x00\x00\x02\x00isomiso2mp41\x00\x00\x00\x08free\
↪x00\x01\x9d\xffmdat\x00...
```

12.8 Dropping an User-Defined Function (UDF)

```
cursor.execute("DROP UDF HFSegmentation;")
response = cursor.fetch_all()
response.as_df()
```

```
                                          0
0  UDF HFSegmentation successfully dropped
```

CHAPTER

THIRTEEN

CHATGPT TUTORIAL

13.1 Start EVA server

We are reusing the start server notebook for launching the EVA server

```
!wget -nc "https://raw.githubusercontent.com/georgia-tech-db/eva/master/tutorials/00-
↪start-eva-server.ipynb"
%run 00-start-eva-server.ipynb

cursor = connect_to_server()
```

```
File '00-start-eva-server.ipynb' already there; not retrieving.
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

```
nohup eva_server > eva.log 2>&1 &
```

```
[notice] A new release of pip is available: 23.0.1 -> 23.1.2
[notice] To update, run: pip install --upgrade pip
```

```
Note: you may need to restart the kernel to use updated packages.
```

13.2 Download News Video and ChatGPT UDF

```
# Download News Video
!wget -nc "https://www.dropbox.com/s/rfm1kds2mv77pca/russia_ukraine.mp4?dl=0" -O russia_
↪ukraine.mp4

# Download ChatGPT UDF if needed
!wget -nc https://raw.githubusercontent.com/georgia-tech-db/eva/master/eva/udfs/chatgpt.
↪py -O chatgpt.py
```

```
File 'russia_ukraine.mp4' already there; not retrieving.
```

```
File ‘chatgpt.py’ already there; not retrieving.
```

13.3 Visualize Video

```
from IPython.display import Video
Video("russia_ukraine.mp4", height=450, width=800, embed=True)
```

```
<IPython.core.display.Video object>
```

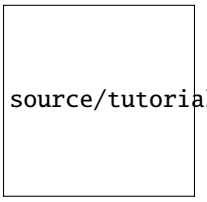
13.4 Set your OpenAI API key here

```
from eva.configuration.configuration_manager import ConfigurationManager
import os

# Assuming that the key is stored as an environment variable
open_ai_key = os.environ.get('OPENAI_KEY')

ConfigurationManager().update_value("third_party", "openai_api_key", open_ai_key)
```

```
# Drop the UDF if it already exists
drop_udf_query = f"DROP UDF IF EXISTS ChatGPT;"
cursor.execute(drop_udf_query)
response = cursor.fetch_all()
response.as_df()

# Register the ChatGPT UDF in EVA
create_udf_query = f"""CREATE UDF ChatGPT
                       IMPL 'chatgpt.py' """
cursor.execute(create_udf_query)
response = cursor.fetch_all()
response.as_df()
```

```
                                                  0
0  UDF ChatGPT successfully added to the database.
```

13.5 Run the ChatGPT UDF



```
#load the video
cursor.execute("LOAD VIDEO 'russia_ukraine.mp4' INTO VIDEOS;")
response = cursor.fetch_all()
response.as_df()
```

```
                            0
0  Number of loaded VIDEO: 1
```

```
# Drop the Text Summarization UDF if needed
cursor.execute("DROP UDF IF EXISTS SpeechRecognizer;")
response = cursor.fetch_all()
response.as_df()

# Create a Text Summarization UDF using Hugging Face
text_summarizer_udf_creation = """
        CREATE UDF SpeechRecognizer
        TYPE HuggingFace
        'task' 'automatic-speech-recognition'
        'model' 'openai/whisper-base';
        """
cursor.execute(text_summarizer_udf_creation)
response = cursor.fetch_all()
response.as_df()
```

```
                                                   0
0  UDF SpeechRecognizer successfully added to the...
```

```
# Drop the table if needed
cursor.execute("DROP TABLE IF EXISTS TEXT_SUMMARY;")
response = cursor.fetch_all()
response.as_df()


# Create a materialized view of the text summarization output
text_summarization_query = """
    CREATE MATERIALIZED VIEW
    TEXT_SUMMARY(text) AS
    SELECT SpeechRecognizer(audio) FROM VIDEOS;
    """
cursor.execute(text_summarization_query)
response = cursor.fetch_all()
response.as_df()
```

```
Empty DataFrame
Columns: []
Index: []
```

```
# Run ChatGPT over the Text Summary extracted by Whisper
chatgpt_udf = """
      SELECT ChatGPT('Is this video summary related to Ukraine russia war',text)
      FROM TEXT_SUMMARY;
```

(continues on next page)

(continued from previous page)

```
    """
cursor.execute(chatgpt_udf)
response = cursor.fetch_all()
response.as_df()
```

```
                                    chatgpt.response
0  No, this video summary is not related to the U...
1  Yes, the video summary is related to the Ukrai...
```

13.6 Check if it works on an SNL Video

```
# Download Entertainment Video
!wget -nc "https://www.dropbox.com/s/u66im8jw2s1dmuw/snl.mp4?dl=0" -O snl.mp4

cursor.execute("DROP TABLE IF EXISTS SNL_VIDEO;")
response = cursor.fetch_all()
response.as_df()

cursor.execute("LOAD VIDEO 'snl.mp4' INTO SNL_VIDEO;")
response = cursor.fetch_all()
response.as_df()
```

```
File ‘snl.mp4’ already there; not retrieving.
```

```
                           0
0  Number of loaded VIDEO: 1
```

```
from IPython.display import Video
Video("snl.mp4", height=450, width=800, embed=True)
```

```
<IPython.core.display.Video object>
```

```
# Drop the table if needed
cursor.execute("DROP TABLE IF EXISTS SNL_TEXT_SUMMARY;")
response = cursor.fetch_all()
response.as_df()


# Create a materialized view of the text summarization output
text_summarization_query = """
    CREATE MATERIALIZED VIEW
    SNL_TEXT_SUMMARY(text) AS
    SELECT SpeechRecognizer(audio) FROM SNL_VIDEO;
    """
cursor.execute(text_summarization_query)
response = cursor.fetch_all()
response.as_df()
```

```
Empty DataFrame
Columns: []
Index: []
```

13.6.1 ChatGPT: Is this video summary related to Ukraine War?

```
# Run ChatGPT over the Text Summary extracted by Whisper
chatgpt_udf = """
      SELECT ChatGPT('Is this video summary related to Ukraine russia war',text)
      FROM SNL_TEXT_SUMMARY;
      """
cursor.execute(chatgpt_udf)
response = cursor.fetch_all()
response.as_df()
```

```
                                    chatgpt.response
0  No, this video summary is not related to the U...
```

13.6.2 ChatGPT: Is this video summary related to a hospital?

```
# Run ChatGPT over the Text Summary extracted by Whisper
chatgpt_udf = """
      SELECT ChatGPT('Is this video summary related to a hospital',text)
      FROM SNL_TEXT_SUMMARY;
      """
cursor.execute(chatgpt_udf)
response = cursor.fetch_all()
response.as_df()
```

```
                                    chatgpt.response
0  Yes, the video summary is related to a hospita...
```

CHAPTER

FOURTEEN

EVA QUERY LANGUAGE REFERENCE

EVA Query Language (EVAQL) is derived from SQL. It is tailored for video analytics. EVAQL allows users to invoke deep learning models in the form of user-defined functions (UDFs).

Here is an example where we first define a UDF wrapping around the FastRCNN object detection model. We then issue a query with this function to detect objects.

```
--- Create an user-defined function wrapping around FastRCNN ObjectDetector
CREATE UDF IF NOT EXISTS FastRCNNObjectDetector
INPUT  (frame NDARRAY UINT8(3, ANYDIM, ANYDIM))
OUTPUT (labels NDARRAY STR(ANYDIM), bboxes NDARRAY FLOAT32(ANYDIM, 4),
        scores NDARRAY FLOAT32(ANYDIM))
TYPE  Classification
IMPL  'eva/udfs/fastrcnn_object_detector.py';

--- Use the function to retrieve frames that contain more than 3 cars
SELECT id FROM MyVideo
WHERE ArrayCount(FastRCNNObjectDetector(data).label, 'car') > 3
ORDER BY id;
```

This page presents a list of all the EVAQL statements that you can leverage in your Jupyter Notebooks.

14.1 LOAD

14.1.1 LOAD VIDEO FROM FILESYSTEM

```
LOAD VIDEO 'test_video.mp4' INTO MyVideo;
```

- **test_video.mp4** is the location of the video file in the filesystem on the client.
- **MyVideo** is the name of the table in EVA where this video is loaded. Subsequent queries over the video must refer to this table name.

When a video is loaded, there is no need to specify the schema for the video table. EVA automatically generates the following schema with two columns: `id` and `data`, that correspond to the frame id and frame content (in Numpy format).

14.1.2 LOAD VIDEO FROM S3

```
LOAD VIDEO 's3://bucket/dummy.avi' INTO MyVideo;
LOAD VIDEO 's3://bucket/eva_videos/*.mp4' INTO MyVideos;
```

The videos are downloaded to a directory that can be configured in the EVA configuration file under *storage:s3_download_dir*. The default directory is `~/.eva/s3_downloads`.

14.1.3 LOAD CSV

To **LOAD** a CSV file, we need to first specify the table schema.

```
CREATE TABLE IF NOT EXISTS MyCSV (
                id INTEGER UNIQUE,
                frame_id INTEGER,
                video_id INTEGER,
                dataset_name TEXT(30),
                label TEXT(30),
                bbox NDARRAY FLOAT32(4),
                object_id INTEGER
            );

LOAD CSV 'test_metadata.csv' INTO MyCSV;
```

- **test_metadata.csv** needs to be loaded onto the server using **LOAD** statement.
- The CSV file may contain additional columns. EVA will only load the columns listed in the defined schema.

14.2 SELECT

14.2.1 SELECT FRAMES WITH PREDICATES

Search for frames with a car

```
SELECT id, frame
FROM MyVideo
WHERE ['car'] <@ FastRCNNObjectDetector(frame).labels
ORDER BY id;
```

Search frames with a pedestrian and a car

```
SELECT id, frame
FROM MyVideo
WHERE ['pedestrian', 'car'] <@ FastRCNNObjectDetector(frame).labels;
```

Search for frames containing greater than 3 cars

```
SELECT id FROM MyVideo
WHERE ArrayCount(FastRCNNObjectDetector(data).label, 'car') > 3
ORDER BY id;
```

14.2.2 SELECT WITH MULTIPLE UDFS

Compose multiple user-defined functions in a single query to construct semantically complex queries.

```
SELECT id, bbox, EmotionDetector(Crop(data, bbox))
FROM HAPPY JOIN LATERAL UNNEST(FaceDetector(data)) AS Face(bbox, conf)
WHERE id < 15;
```

14.3 EXPLAIN

14.3.1 EXPLAIN QUERY

List the query plan associated with a EVAQL query

Append `EXPLAIN` in front of the query to retrieve the plan.

```
EXPLAIN SELECT CLASS FROM TAIPAI;
```

14.4 SHOW

14.4.1 SHOW UDFS

List the registered user-defined functions

```
SHOW UDFS;
```

14.5 CREATE

14.5.1 CREATE TABLE

To create a table, specify the schema of the table.

```
CREATE TABLE IF NOT EXISTS MyCSV (
                id INTEGER UNIQUE,
                frame_id INTEGER,
                video_id INTEGER,
                dataset_name TEXT(30),
                label TEXT(30),
                bbox NDARRAY FLOAT32(4),
                object_id INTEGER
 );
```

14.5.2 CREATE UDF

To register an user-defined function, specify the implementation details of the UDF.

```
CREATE UDF IF NOT EXISTS FastRCNNObjectDetector
INPUT  (frame NDARRAY UINT8(3, ANYDIM, ANYDIM))
OUTPUT (labels NDARRAY STR(ANYDIM), bboxes NDARRAY FLOAT32(ANYDIM, 4),
        scores NDARRAY FLOAT32(ANYDIM))
TYPE  Classification
IMPL  'eva/udfs/fastrcnn_object_detector.py';
```

14.5.3 CREATE MATERIALIZED VIEW

To create a view with materialized results – like the outputs of deep learning model, use the following template:

```
CREATE MATERIALIZED VIEW UADETRAC_FastRCNN (id, labels) AS
SELECT id, FastRCNNObjectDetector(frame).labels
FROM UADETRAC
WHERE id<5;
```

14.6 DROP

14.6.1 DROP TABLE

```
DROP TABLE DETRACVideo;
```

14.6.2 DROP UDF

```
DROP UDF FastRCNNObjectDetector;
```

14.7 INSERT

14.7.1 TABLE MyVideo

MyVideo Table schema

```
CREATE TABLE MyVideo
(id INTEGER,
data NDARRAY FLOAT32(ANYDIM));
```

14.7.2 INSERT INTO TABLE

Insert a tuple into a table.

```
INSERT INTO MyVideo (id, data) VALUES
    (1,
        [[[40, 40, 40] , [40, 40, 40]],
         [[40, 40, 40] , [40, 40, 40]]]);
```

14.8 DELETE

14.8.1 DELETE INTO TABLE

Delete a tuple from a table based on a predicate.

```
DELETE FROM MyVideo WHERE id<10;
```

14.9 RENAME

14.9.1 RENAME TABLE

```
RENAME TABLE MyVideo TO MyVideo1;
```

CHAPTER

FIFTEEN

USER-DEFINED FUNCTIONS

This section provides an overview of how you can create and use a custom user-defined function (UDF) in your queries. For example, you could write an UDF that wraps around your custom PyTorch model.

15.1 Part 1: Writing a custom UDF

During each step, use this UDF implementation as a reference.

1. Create a new file under *udfs/* folder and give it a descriptive name. eg: *yolo_object_detection.py*.

   Note: UDFs packaged along with EVA are located inside the udfs folder.

2. Create a Python class that inherits from *PytorchClassifierAbstractUDF*.

- The *PytorchClassifierAbstractUDF* is a parent class that defines and implements standard methods for model inference.
- The functions setup and forward should be implemented in your child class. These functions can be implemented with the help of Decorators.

15.2 Setup

An abstract method that must be implemented in your child class. The setup function can be used to initialize the parameters for executing the UDF. The parameters that need to be set are

- cacheable: bool
  - True: Cache should be enabled. Cache will be automatically invalidated when the UDF changes.
  - False: cache should not be enabled.
- udf_type: str
  - object_detection: UDFs for object detection.
- batchable: bool
  - True: Batching should be enabled
  - False: Batching is disabled.

The custom setup operations for the UDF can be written inside the function in the child class. If there is no need for any custom logic, then you can just simply write "pass" in the function definition.

Example of a Setup function

```
@setup(cachable=True, udf_type="object_detection", batchable=True)
def setup(self, threshold=0.85):
    #custom setup function that is specific for the UDF
    self.threshold = threshold
    self.model = torch.hub.load("ultralytics/yolov5", "yolov5s", verbose=False)
```

15.3 Forward

An abstract method that must be implemented in your UDF. The forward function receives the frames and runs the deep learning model on the data. The logic for transforming the frames and running the models must be provided by you. The arguments that need to be passed are

- input_signatures: List[IOColumnArgument]

  Data types of the inputs to the forward function must be specified. If no constraints are given, then no validation is done for the inputs.

- output_signatures: List[IOColumnArgument]

  Data types of the outputs to the forward function must be specified. If no constraints are given, then no validation is done for the inputs.

A sample forward function is given below

```
@forward(
      input_signatures=[
          PyTorchTensor(
              name="input_col",
              is_nullable=False,
              type=NdArrayType.FLOAT32,
              dimensions=(1, 3, 540, 960),
          )
      ],
      output_signatures=[
          PandasDataframe(
              columns=["labels", "bboxes", "scores"],
              column_types=[
                  NdArrayType.STR,
                  NdArrayType.FLOAT32,
                  NdArrayType.FLOAT32,
              ],
              column_shapes=[(None,), (None,), (None,)],
          )
      ],
  )
  def forward(self, frames: Tensor) -> pd.DataFrame:
    #the custom logic for the UDF
    outcome = []
```

(continues on next page)

(continued from previous page)

```
    frames = torch.permute(frames, (0, 2, 3, 1))
    predictions = self.model([its.cpu().detach().numpy() * 255 for its in frames])

    for i in range(frames.shape[0]):
        single_result = predictions.pandas().xyxy[i]
        pred_class = single_result["name"].tolist()
        pred_score = single_result["confidence"].tolist()
        pred_boxes = single_result[["xmin", "ymin", "xmax", "ymax"]].apply(
            lambda x: list(x), axis=1
        )

        outcome.append(
            {"labels": pred_class, "bboxes": pred_boxes, "scores": pred_score}
        )

    return pd.DataFrame(outcome, columns=["labels", "bboxes", "scores"])
```

15.4 Part 2: Registering and using the UDF in EVA Queries

Now that you have implemented your UDF, we need to register it as a UDF in EVA. You can then use the UDF in any query.

1. Register the UDF with a query that follows this template:

   CREATE UDF [IF NOT EXISTS] <name> IMPL <path_to_implementation>;

   where,

   - <name> - specifies the unique identifier for the UDF.
   - <path_to_implementation> - specifies the path to the implementation class for the UDF

   Here, is an example query that registers a UDF that wraps around the 'YoloObjectDetection' model that performs Object Detection.

   ```
   CREATE UDF YoloDecorators
   IMPL  'eva/udfs/decorators/yolo_object_detection_decorators.py';
   ```

   A status of 0 in the response denotes the successful registration of this UDF.

2. Now you can execute your UDF on any video:

   ```
   SELECT YoloDecorators(data) FROM MyVideo WHERE id < 5;
   ```

3. You can drop the UDF when you no longer need it.

   ```
   DROP UDF IF EXISTS YoloDecorators;
   ```

15.5 Ultralytics Models

This section provides an overview of how you can use out-of-the-box Ultralytics models in EVA.

15.5.1 Creating YOLO Model

To create a YOLO UDF in EVA using Ultralytics models, use the following SQL command:

```
CREATE UDF IF NOT EXISTS Yolo
TYPE ultralytics
'model' 'yolov8m.pt'
```

You can change the *model* value to specify any other model supported by Ultralytics.

15.5.2 Supported Models

The following models are currently supported by Ultralytics in EVA:

- yolov8n.pt
- yolov8s.pt
- yolov8m.pt
- yolov8l.pt
- yolov8x.pt

Please refer to the Ultralytics documentation for more information about these models and their capabilities.

15.5.3 Using Ultralytics Models with Other UDFs

This code block demonstrates how the YOLO model can be combined with other models such as Color and DogBreedClassifier to perform more specific and targeted object detection tasks. In this case, the goal is to find images of black-colored Great Danes.

The first query uses YOLO to detect all images of dogs with black color. The `UNNEST` function is used to split the output of the `Yolo` UDF into individual rows, one for each object detected in the image. The `Color` UDF is then applied to the cropped portion of the image to identify the color of each detected dog object. The `WHERE` clause filters the results to only include objects labeled as "dog" and with a color of "black".

```
SELECT id, bbox FROM dogs
JOIN LATERAL UNNEST(Yolo(data)) AS Obj(label, bbox, score)
WHERE Obj.label = 'dog'
AND Color(Crop(data, bbox)) = 'black';
```

The second query builds upon the first by further filtering the results to only include images of Great Danes. The `DogBreedClassifier` UDF is used to classify the cropped portion of the image as a Great Dane. The `WHERE` clause adds an additional condition to filter the results to only include objects labeled as "dog", with a color of "black", and classified as a "great dane".

```
SELECT id, bbox FROM dogs
JOIN LATERAL UNNEST(Yolo(data)) AS Obj(label, bbox, score)
WHERE Obj.label = 'dog'
```

(continues on next page)

(continued from previous page)

```
AND DogBreedClassifier(Crop(data, bbox)) = 'great dane'
AND Color(Crop(data, bbox)) = 'black';
```

15.6 HuggingFace Models

This section provides an overview of how you can use out-of-the-box HuggingFace models in EVA.

15.6.1 Creating UDF from HuggingFace

EVA supports UDFS similar to Pipelines in HuggingFace.

```
CREATE UDF IF NOT EXISTS HFObjectDetector
TYPE  HuggingFace
'task' 'object-detection'
'model' 'facebook / detr-resnet-50'
```

EVA supports all arguments supported by HF pipelines. You can pass those using a key value format similar to task and model above.

15.6.2 Supported Tasks

EVA supports the following tasks from huggingface:

- Audio Classification
- Automatic Speech Recognition
- Text Classification
- Summarization
- Text2Text Generation
- Text Generation
- Image Classification
- Image Segmentation
- Image-to-Text
- Object Detection
- Depth Estimation

15.7 OpenAI Models

This section provides an overview of how you can use OpenAI models in EVA.

15.7.1 Chat Completion UDFs

To create a chat completion UDF in EVA, use the following SQL command:

```
CREATE UDF IF NOT EXISTS OpenAIChatCompletion
IMPL 'eva/udfs/openai_chat_completion_udf.py'
'model' 'gpt-3.5-turbo'
```

EVA supports the following models for chat completion task:

- "gpt-4"
- "gpt-4-0314"
- "gpt-4-32k"
- "gpt-4-32k-0314"
- "gpt-3.5-turbo"
- "gpt-3.5-turbo-0301"

The chat completion UDF can be composed in interesting ways with other UDFs. Please refer to the ChatGPT notebook for an example of combining chat completion task with caption extraction and video summarization models from Hugging Face and feeding it to chat completion to ask questions about the results.

15.8 User-Defined Functions

This section provides an overview of how you can create and use a custom user-defined function (UDF) in your queries. For example, you could write a UDF that wraps around a PyTorch model.

15.8.1 Part 1: Writing a Custom UDF

During each step, use the UDF implementation as a reference.

1. Create a new file under *udfs/* folder and give it a descriptive name, e.g., *fastrcnn_object_detector.py*.

   Note: UDFs packaged along with EVA are located inside the udfs folder.

2. Create a Python class that inherits from *PytorchClassifierAbstractUDF*.
   - The `PytorchClassifierAbstractUDF` is a parent class that defines and implements standard methods for model inference.
   - Implement the `setup` and `forward` functions in your child class. These functions can be implemented with the help of decorators.

15.8.2 Setup

An abstract method that must be implemented in your child class. The *setup* function can be used to initialize the parameters for executing the UDF. The following parameters must be set:

- `cacheable`: bool
  - *True*: Cache should be enabled. The cache will be automatically invalidated when the UDF changes.
  - *False*: Cache should not be enabled.
- `udf_type`: str
  - *object_detection*: UDFs for object detection.
- `batchable`: bool
  - *True*: Batching should be enabled.
  - *False*: Batching is disabled.

The custom setup operations for the UDF can be written inside the function in the child class. If no custom logic is required, then you can just write *pass* in the function definition.

Example of the *setup* function:

```
@setup(cachable=True, udf_type="object_detection", batchable=True)
def setup(self, threshold=0.85):
    self.threshold = threshold
    self.model = torchvision.models.detection.fasterrcnn_resnet50_fpn(
        weights="COCO_V1", progress=False
    )
    self.model.eval()
```

In this instance, we have configured the *cachable* and *batchable* attributes to *True*. As a result, EVA will cache the UDF outputs and utilize batch processing for increased efficiency.

15.8.3 Forward

An abstract method that must be implemented in your child class. The *forward* function receives the frames and runs the Deep Learning model on the frames. The logic for transforming the frames and running the models must be provided by you. The arguments that need to be passed are:

- `input_signatures`: List[IOColumnArgument]

  Data types of the inputs to the *forward* function must be specified. If no constraints are given, no validation is done for the inputs.

- `output_signatures`: List[IOColumnArgument]

  Data types of the outputs from the *forward* function must be specified. If no constraints are given, no validation is done for the inputs.

A sample `forward` function is given below:

```
@forward(
    input_signatures=[
        PyTorchTensor(
            name="input_col",
            is_nullable=False,
```

(continues on next page)

(continued from previous page)

```
            type=NdArrayType.FLOAT32,
            dimensions=(1, 3, 540, 960),
        )
    ],
    output_signatures=[
        PandasDataframe(
            columns=["labels", "bboxes", "scores"],
            column_types=[
                NdArrayType.STR,
                NdArrayType.FLOAT32,
                NdArrayType.FLOAT32,
            ],
            column_shapes=[(None,), (None,), (None,)],
        )
    ],
)
def forward(self, frames: Tensor) -> pd.DataFrame:
    predictions = self.model(frames)
    outcome = []
    for prediction in predictions:
        pred_class = [
            str(self.labels[i]) for i in list(self.as_numpy(prediction["labels"]))
        ]
        pred_boxes = [
            [i[0], i[1], i[2], i[3]]
            for i in list(self.as_numpy(prediction["boxes"]))
        ]
```

In this instance, the forward function takes a PyTorch tensor of Float32 type with a shape of (1, 3, 540, 960) as input. The resulting output is a pandas dataframe with 3 columns, namely "labels", "bboxes", and "scores", and of string, float32, and float32 types respectively.

15.8.4 Part 2: Registering and using the UDF in queries

Now that you have implemented your UDF we need to register it in EVA. You can then use the function in any query.

Register the UDF in EVA

```
CREATE UDF [ IF NOT EXISTS ] <name>
IMPL <implementation_path>;
```

`name` - specifies the unique identifier for the UDF.

`implementation_path` - specifies the path to the implementation class for the UDF

Here, is an example query that registers a UDF that wraps around the `fasterrcnn_resnet50_fpn` model that performs Object Detection.

```
CREATE UDF FastrcnnObjectDetector
IMPL  'eva/udfs/fastrcnn_object_detector.py';
```

Call registered UDF in a query

```
SELECT FastrcnnObjectDetector(data) FROM MyVideo WHERE id < 5;
```

Drop the UDF

```
DROP UDF IF EXISTS FastrcnnObjectDetector;
```

CHAPTER

SIXTEEN

IO DESCRIPTORS

EVA supports three key data types. The inputs and outputs of the user-defined functions (UDFs) must be of one of these types.

16.1 NumpyArray

Used when the inputs or outputs of the UDF is of type Numpy Array.

16.2 Parameters

name (*str*): name of the numpy array.

is_nullable (*bool*): boolean value indicating if the numpy array can be NULL.

type (*NdArrayType*): data type of all the elements in the numpy array. The available types can be found in eva/catalog/catalog_type.py in the class *NdArrayType*

dimensions(*Tuple(int)*): shape of the numpy array

```
from eva.catalog.catalog_type import NdArrayType
NumpyArray(
        name="input_arr",
        is_nullable=False,
        type=NdArrayType.INT32,
        dimensions=(2, 2),
)
```

16.3 PyTorchTensor

name (*str*): name of the pytorch tensor.

is_nullable (*bool*): boolean value indicating if the pytorch tensor can be NULL.

type (*NdArrayType*): data type of elements in the pytorch tensor. The available types can be found in eva/catalog/catalog_type.py in class NdArrayType

dimensions(*Tuple(int)*): shape of the numpy array

```
from eva.catalog.catalog_type import NdArrayType
PyTorchTensor(
                name="input_arr",
                is_nullable=False,
                type=NdArrayType.INT32,
                dimensions=(2, 2),
)
```

16.4 PandasDataframe

columns (*List[str]*): list of strings that represent the expected column names in the pandas dataframe that is returned from the UDF.

column_types (*NdArrayType*): expected datatype of the column in the pandas dataframe returned from the UDF. The NdArrayType class is inherited from eva.catalog.catalog_type.

column_shapes (*List[tuples]*): list of tuples that represent the expected shapes of columns in the pandas dataframe returned from the UDF.

```
PandasDataframe(
            columns=["labels", "bboxes", "scores"],
            column_types=[
                NdArrayType.STR,
                NdArrayType.FLOAT32,
                NdArrayType.FLOAT32,
            ],
            column_shapes=[(None,), (None,), (None,)],
)
```

CHAPTER

SEVENTEEN

CONFIGURE GPU

1. Queries in EVA use deep learning models that run much faster on a GPU as opposed to a CPU. If your workstation has a GPU, you can configure EVA to use the GPU during query execution. Use the following command to check your hardware capabilities:

```
ubuntu-drivers devices
nvidia-smi
```

A valid output from the command indicates that your GPU is configured and ready to use. If not, you will need to install the appropriate GPU driver. This page provides a step-by-step guide on installing and configuring the GPU driver in the Ubuntu Operating System.

- When installing an NVIDIA driver, ensure that the version of the GPU driver is correct to avoid compatibiility issues.
- When installing cuDNN, you will need to create an account and ensure that you get the correct *deb* files for your operating system and architecture.

2. You can run the following code in a Jupyter notebook to verify that your GPU is detected by PyTorch:

```
import torch
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

Output of *cuda:0* indicates the presence of a GPU. 0 indicates the index of the GPU in system. If you have multiple GPUs on your workstation, the index must be updated accordingly.

3. Now configure the executor section in eva.yml as follows:

```
executor:
    gpus: {'127.0.1.1': [0]}
```

Here, *127.0.1.1* is the loopback address on which the EVA server is running. 0 refers to the GPU index to be used.

CHAPTER

EIGHTEEN

EVA INTERNALS

18.1 Path of a Query

The following code represents a sequence of operations that can be used to execute a query in a evaql database. found in eva/server/command_handler.py

> Parse the query using the Parser() function provided by the evaql library. The result of this step will be a parsed representation of the query in the form of an abstract syntax tree (AST).

```
stmt = Parser().parse(query)[0]
```

Bind the parsed AST to a statement context using the StatementBinder() function. This step resolves references to schema objects and performs other semantic checks on the query.

```
StatementBinder(StatementBinderContext()).bind(stmt)
```

Convert the bound AST to a logical plan using the StatementToPlanConvertor() function. This step generates a logical plan that specifies the sequence of operations needed to execute the query.

```
l_plan = StatementToPlanConvertor().visit(stmt)
```

Generate a physical plan from the logical plan using the plan_generator.build() function. This step optimizes the logical plan and generates a physical plan that specifies how the query will be executed.

```
p_plan = plan_generator.build(l_plan)
```

Execute the physical plan using the PlanExecutor() function. This step retrieves the data from the database and produces the final output of the query.

```
output = PlanExecutor(p_plan).execute_plan()
```

Overall, this sequence of operations represents the path of query execution in a evaql database, from parsing the query to producing the final output.

18.2 Topics

18.2.1 Catalog

Catalog Manager

Explanation for developers on how to use the eva catalog_manager.

CatalogManager class that provides a set of services to interact with a database that stores metadata about tables, columns, and user-defined functions (UDFs). Information like what is the data type in a certain column in a table, type of a table, its name, etc.. It contains functions to get, insert and delete catalog entries for Tables, UDFs, UDF IOs, Columns and Indexes.

This data is stored in the eva_catalog.db file which can be found in *~/.eva/<version>/* folder.

Catalog manager currently has 5 services in it:

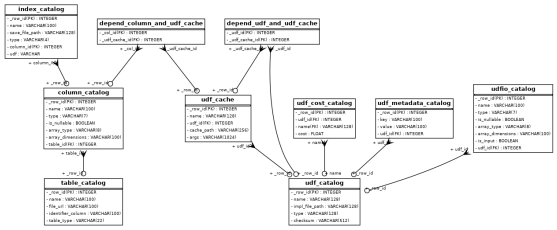
```
TableCatalogService()
ColumnCatalogService()
UdfCatalogService()
UdfIOCatalogService()
IndexCatalogService()
```

Catalog Services

This class provides functionality related to a table catalog, including inserting, getting, deleting, and renaming table entries, as well as retrieving all entries. e.g. the TableCatalogService contains code to get, insert and delete a table.

Catalog Models

These contain the data model that is used by the catalog services. Each model represents a table in the underlying database.



CHAPTER

NINETEEN

CONTRIBUTING

We welcome all kinds of contributions to EVA.

- Code reviews
- Improving documentation
- Tutorials and applications
- New features

19.1 Setting up the Development Environment

First, you will need to checkout the repository from GitHub and build EVA from the source. Follow the following instructions to build EVA locally. We recommend using a virtual environment and the pip package manager.

```
git clone https://github.com/georgia-tech-db/eva.git && cd eva
python3 -m venv test_eva_db       # create a virtual environment
source test_eva_db/bin/activate   # activate the virtual environment
pip install --upgrade pip         # upgrade pip
pip install -e ".[dev]"           # build and install the EVA package
bash script/test/test.sh          # run the eva EVA suite
```

After installing the package locally, you can make changes and run the test cases to check their impact.

```
pip install .          # reinstall EVA package to include local changes
pkill -9 eva_server    # kill running EVA server (if any)
eva_server&            # launch EVA server with newly installed package
```

19.2 Testing

Check if your local changes broke any unit or integration tests by running the following script:

```
bash script/test/test.sh
```

If you want to run a specific test file, use the following command.

```
python -m pytest test/integration_tests/test_select_executor.py
```

Use the following command to run a specific test case within a specific test file.

```
python -m pytest test/integration_tests/test_select_executor.py -k 'test_should_load_and_
↪select_in_table'
```

19.3 Submitting a Contribution

Follow the following steps to contribute to EVA:

- Merge the most recent changes from the master branch

```
git remote add origin git@github.com:georgia-tech-db/eva.git
git pull . origin/master
```

- Run the *test script* to ensure that all the test cases pass.
- If you are adding a new EVAQL command, add an illustrative example usage in the documentation.
- Run the following command to ensure that code is properly formatted.

```
python script/formatting/formatter.py
```

19.4 Code Style

We use the black code style for formatting the Python code. For docstrings and documentation, we use Google Pydoc format.

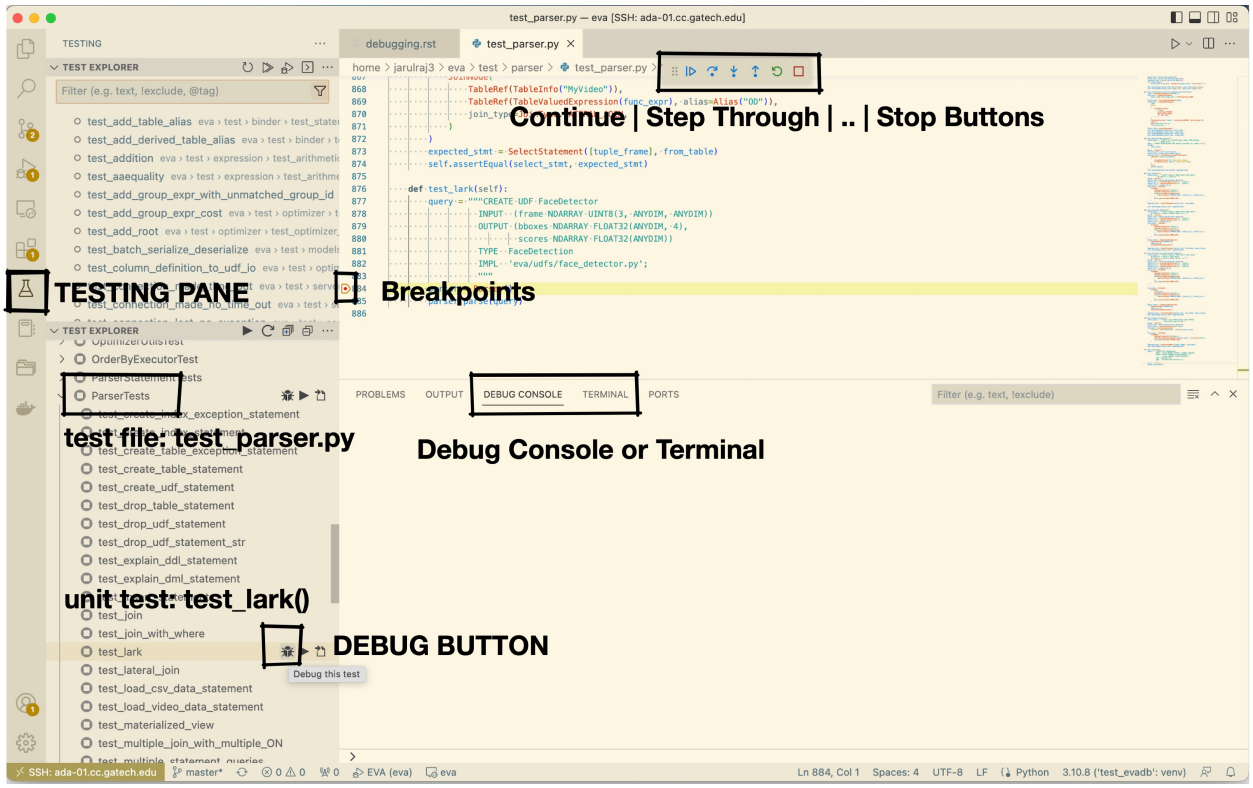
```
def function_with_types_in_docstring(param1, param2) -> bool:
    """Example function with types documented in the docstring.

    Additional explanatory text can be added in paragraphs.

    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.

    Returns:
        bool: The return value. True for success, False otherwise.
```

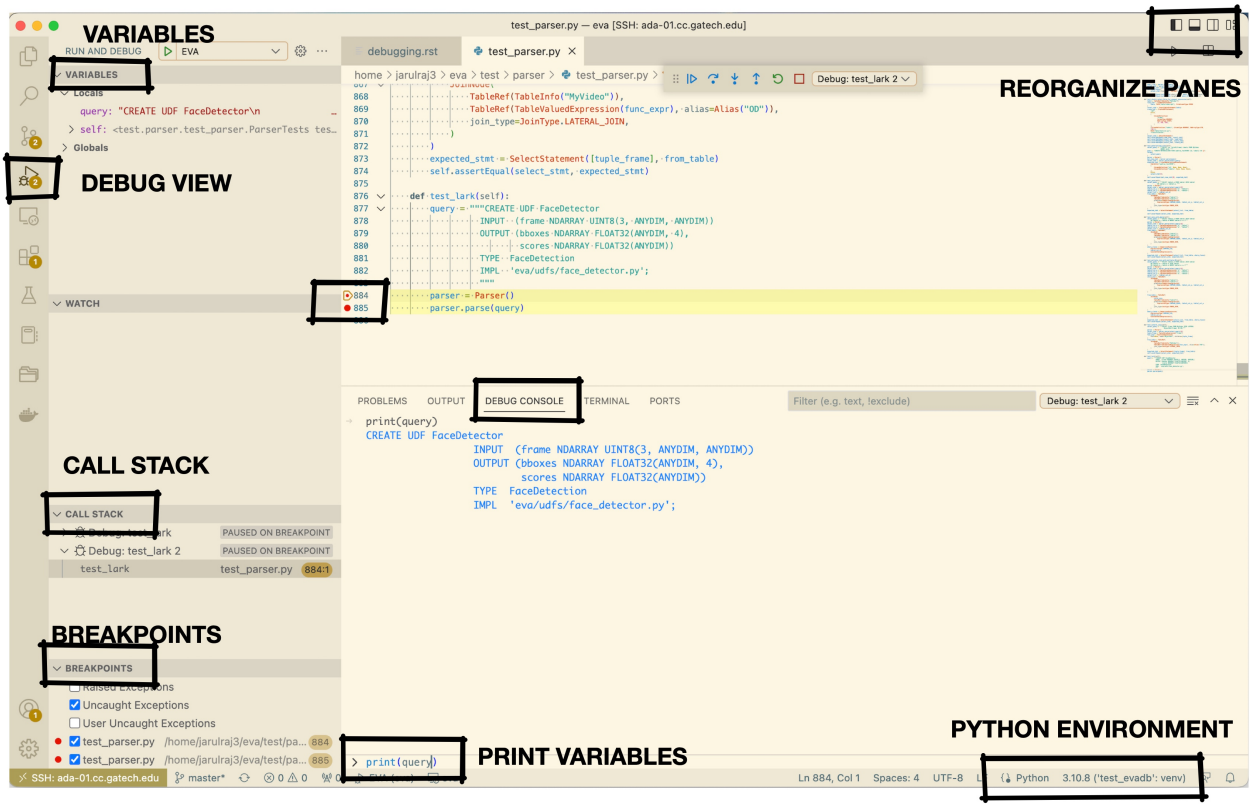
19.5 Debugging

We recommend using Visual Studio Code with a debugger for developing EVA. Here are the steps for setting up the development environment:

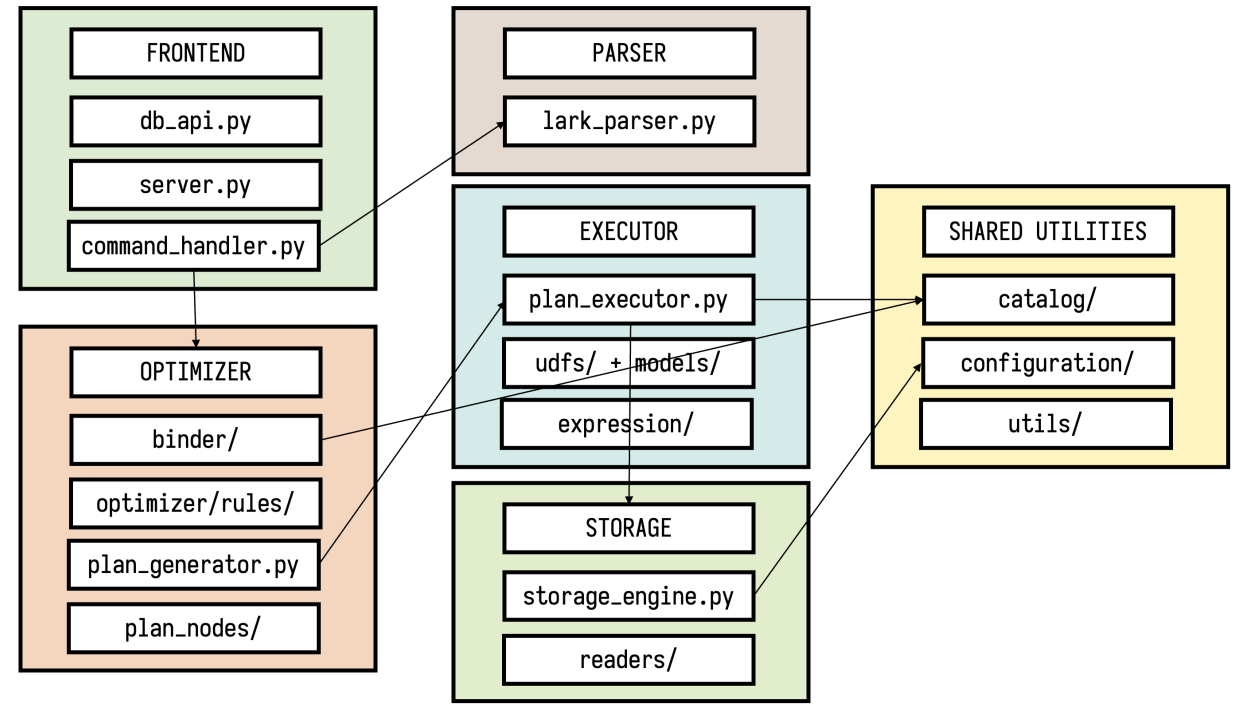
1. Install the Python extension in Visual Studio Code.
2. Install the Python Test Explorer extension.

3. Follow these instructions to run a particular test case from the file: Getting started.


Continue | Step Through | .. | Stop Buttons
TESTING PANE
Breakpoints
test file: test_parser.py
Debug Console or Terminal
unit test: test_lark()
DEBUG BUTTON


VARIABLES
REORGANIZE PANES
DEBUG VIEW
CALL STACK
BREAKPOINTS
PRINT VARIABLES
PYTHON ENVIRONMENT

19.6 Architecture Diagram



19.7 Troubleshooting

If the test suite fails with a *PermissionDenied* exception, update the *path_prefix* attribute under the *storage* section in the EVA configuration file (`~/.eva/eva.yml`) to a directory where you have write privileges.

CHAPTER

TWENTY

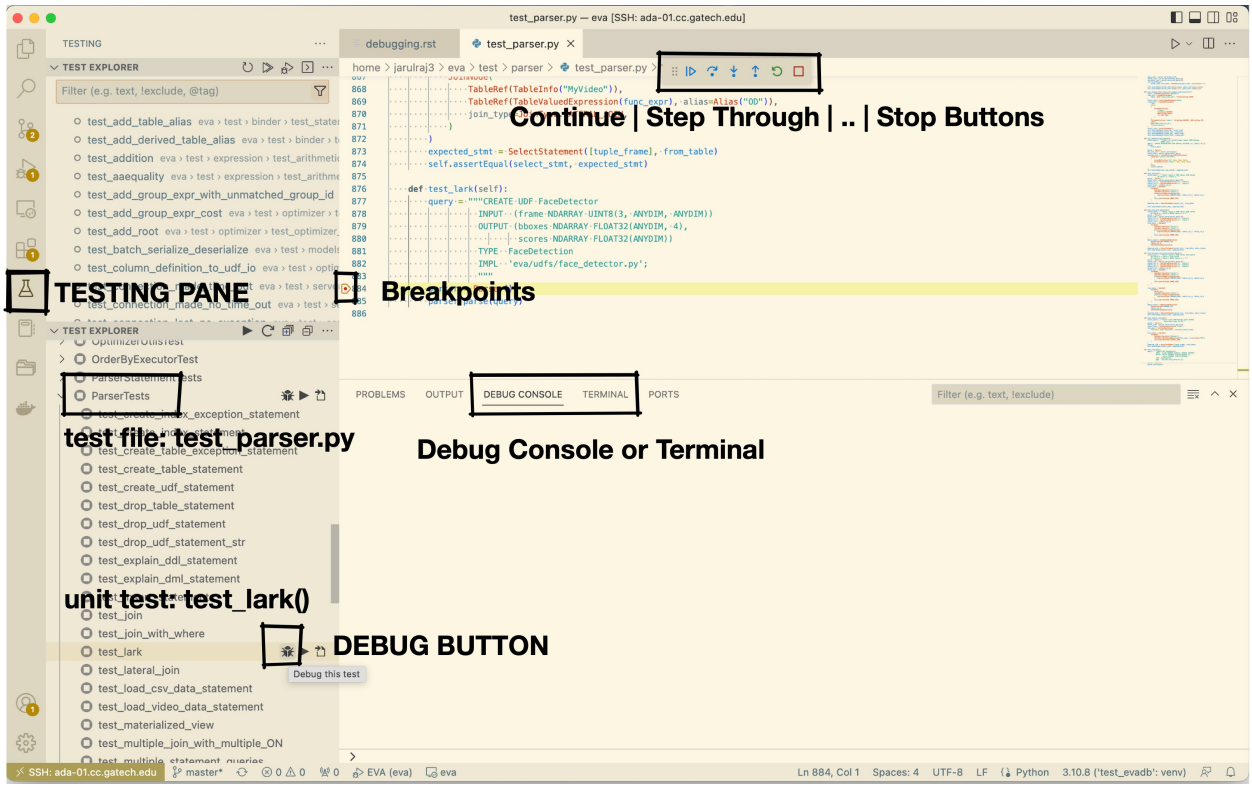
DEBUGGING

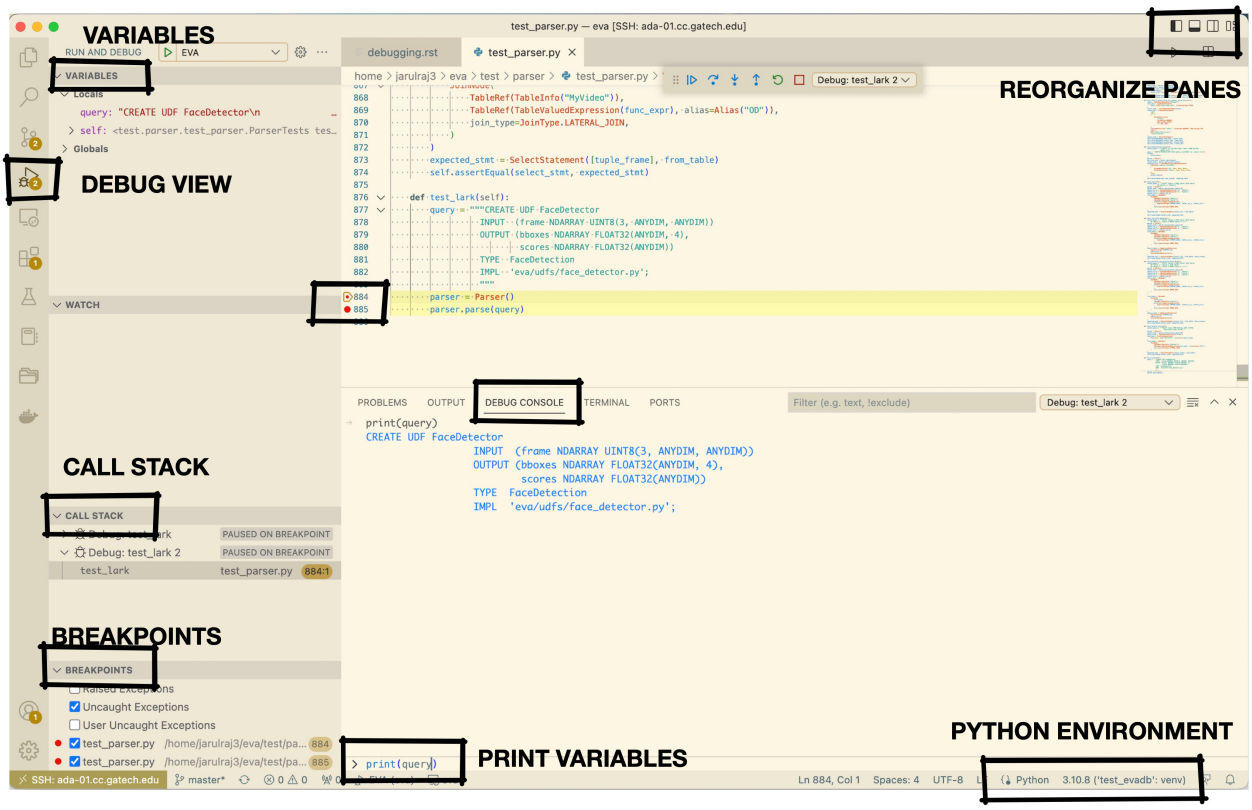
We recommend Visual Studio Code with a debugger for debugging EVA. This tutorial presents a detailed step-by-step process of using the debugger.

20.1 Setup debugger

1. Install the Python extension in Visual Studio Code.
2. Install the Python Test Explorer extension.

3. Follow these instructions to run a particular test case from the file: Getting started.





20.2 Alternative: Manually Setup Debugger for EVA

When you press the debug icon, you will be given an option to create a `launch.json` file.

While creating the JSON file, you will be prompted to select the environment to be used. Select the python environment from the `command palette` at the top. If the Python environment cannot be seen in the drop-down menu, try installing the python extension, and repeat the process.

Once you select the python environment, a `launch.json` file will be created with the default configurations set to debug a simple .py file.

More configurations can further be added to the file, to modify the environment variables or to debug an entire folder or workspace directory. Use the following configuration in the JSON file:

```
{

    "version": "0.2.0",
    "configurations": [

        {
            "name": "Python: test_pytorch.py",
            "type": "python",
            "request": "launch",
            "program": "${workspaceFolder}/test/integration_tests/test_pytorch.py",
            "console": "integratedTerminal",
            "cwd": "${workspaceFolder}",
            "env": {"PYTHONPATH": "${workspaceRoot}"}
```

(continues on next page)

(continued from previous page)

```
        }

    ]
}
```

You can modify the fields of the above JSON file as follows:

`name`: It is the reader-friendly name to appear in the Debug launch configuration dropdown.

`type`: The type of debugger to use for this launch configuration.

`program`: The executable or file to run when launching the debugger. In the above example, test_integration.py will be executed by the debugger.

`env`: Here you specify the environment variables. In the above example, the path for the conda environment for Eva has been specified.

Using these configuration variables, you can run the debugger both locally as well as on a remote server.

CHAPTER

TWENTYONE

EXTENDING EVA

This document details the steps involved in adding support for a new operator (or command) in EVA. We illustrate the process using a DDL command.

21.1 Command Handler

An input query string is handled by **Parser**, **StatementTOPlanConvertor**, **PlanGenerator**, and **PlanExecutor**. We discuss each part separately.

```
def execute_query(query) -> Iterator[Batch]:
    """
    Execute the query and return a result generator.
    """
    #1. parser
    stmt = Parser().parse(query)[0]
    #2. statement to logical plan
    l_plan = StatementToPlanConvertor().visit(stmt)
    #3. logical to physical plan
    p_plan = PlanGenerator().build(l_plan)
    #4. parser
    return PlanExecutor(p_plan).execute_plan()
```

21.2 1. Parser

The parser firstly generate **syntax tree** from the input string, and then tansform syntax tree into **statement**.

The first part of Parser is build from a LARK grammar file.

21.2.1 parser/eva

- `eva.lark` - add keywords(eg. CREATE, TABLE) under **Common Keywords**
  - Add new grammar rule (eg. create_table)
  - Write a new grammar, for example:

```
create_table: CREATE TABLE if_not_exists? table_name create_definitions
```

The second part of parser is implemented as **parser visitor**.

21.2.2 parser/lark_visitor

- `_[cmd]_statement.py` - eg. class CreateTable(evaql_parserVisitor)
  - Write functions to transform each input data from syntax tree to desired type. (eg. transform Column information into a list of ColumnDefinition)
  - Write a function to construct [cmd]Statement and return it.
- `__init__.py` - import `_[cmd]_statement.py` and add its class to `ParserVisitor`'s parent class.

```
from src.parser.parser_visitor._create_statement import CenameTable
class ParserVisitor(CommonClauses, CreateTable, Expressions,
                    Functions, Insert, Select, TableSources,
                    Load, Upload):
```

21.2.3 parser/

- `[cmd]_statement.py` - class [cmd]Statement. Its constructor is called in `_[cmd]_statement.py`
- `types.py` - register new StatementType

21.3 2. Statement To Plan Convertor

The part transforms the statement into corresponding logical plan.

21.3.1 Optimizer

- `operators.py`
  - Define class Logical[cmd], which is the logical node for the specific type of command.

```
class LogicalCreate(Operator):
    def __init__(self, video: TableRef, column_list: List[DataFrameColumn], if_not_
↪exists: bool = False, children=None):
    super().__init__(OperatorType.LOGICALCREATE, children)
    self._video = video
    self._column_list = column_list
    self._if_not_exists = if_not_exists
    # ...
```

  - Register new operator type to **class OperatorType**, Notice that must add it **before LOGICALDELIMITER** !!!
- `statement_to_opr_convertor.py`
  - import resource

```
from src.optimizer.operators import LogicalCreate
from src.parser.rename_statement import CreateTableStatement
```

  - implement **visit_[cmd]**() function, which converts statement to operator

```
# May need to convert the statement into another data type.
# The new data type is usable for excuting command.
# For example, column_list -> column_metadata_list

def visit_create(self, statement: AbstractStatement):
    video_ref = statement.table_ref
    if video_ref is None:
        LoggingManager().log("Missing Table Name In Create Statement",
                             LoggingLevel.ERROR)

    if_not_exists = statement.if_not_exists
    column_metadata_list = create_column_metadata(statement.column_list)

    create_opr = LogicalCreate(
        video_ref, column_metadata_list, if_not_exists)
    self._plan = create_opr
```

- modify visit function to call the right visit_[cmd] funciton

```
def visit(self, statement: AbstractStatement):
    if isinstance(statement, SelectStatement):
        self.visit_select(statement)
    #...
    elif isinstance(statement, CreateTableStatement):
        self.visit_create(statement)
    return self._plan
```

21.4 3. Plan Generator

The part transformed logical plan to physical plan. The modified files are stored under **Optimizer** and **Planner** folders.

21.4.1 plan_nodes/

- `[cmd]_plan.py` - class [cmd]Plan, which stored information required for rename table.

```
class CreatePlan(AbstractPlan):
    def __init__(self, video_ref: TableRef,
                 column_list: List[DataFrameColumn],
                 if_not_exists: bool = False):
        super().__init__(PlanOprType.CREATE)
        self._video_ref = video_ref
        self._column_list = column_list
        self._if_not_exists = if_not_exists
    #...
```

- `types.py` - register new plan operator type to PlanOprType

21.4.2 optimizer/rules

- `rules.py`-
  - Import operators
  - Register new ruletype to **RuleType** and **Promise** (place it **before IMPLEMENTATION_DELIMETER** !!)
  - implement class `Logical[cmd]ToPhysical`, its memeber function apply() will construct a corresbonding `[cmd]Plan` object.

```
class LogicalCreateToPhysical(Rule):
    def __init__(self):
    pattern = Pattern(OperatorType.LOGICALCREATE)
    super().__init__(RuleType.LOGICAL_CREATE_TO_PHYSICAL, pattern)

def promise(self):
    return Promise.LOGICAL_CREATE_TO_PHYSICAL

def check(self, before: Operator, context: OptimizerContext):
    return True

def apply(self, before: LogicalCreate, context: OptimizerContext):
    after = CreatePlan(before.video, before.column_list, before.if_not_exists)
    return after
```

- `rules_base.py`-
  - Register new ruletype to **RuleType** and **Promise** (place it **before IMPLEMENTATION_DELIMETER** !!)
- `rules_manager.py`-
  - Import rules created in `rules.py`
  - Add imported logical to physical rules to `self._implementation_rules`

21.5 4. Plan Executor

`PlanExecutor` uses data stored in physical plan to run the command.

21.5.1 executor/

- `[cmd]_executor.py` - implement an executor that make changes in **catalog**, **metadata**, or **storage engine** to run the command.
  - May need to create helper function in CatalogManager, DatasetService, DataFrameMetadata, etc.

```
class CreateExecutor(AbstractExecutor):
    def exec(self):
        if (self.node.if_not_exists):
            # check catalog if we already have this table
            return
```

(continues on next page)

(continued from previous page)

```
        table_name = self.node.video_ref.table_info.table_name
        file_url = str(generate_file_path(table_name))
        metadata = CatalogManager().create_metadata(table_name, file_url, self.node.
↪column_list)

        StorageEngine.create(table=metadata)
```

21.6 Additional Notes

Key data structures in EVA:

- **Catalog**: Records `DataFrameMetadata` for all tables.
  - data stored in DataFrameMetadata: `name`, `file_url`, `identifier_id`, `schema`
    - `file_url` - used to access the real table in storage engine.
  - For the `RENAME` table command, we use the `old_table_name` to access the corresponing entry in metadata table, and the `modified name` of the table.
- **Storage Engine**:
  - API is defined in `src/storage`, currently only supports create, read, write.

CHAPTER

TWENTYTWO

EVA RELEASE GUIDE

22.1 Part 1: Before You Start

Make sure you have PyPI account with maintainer access to the EVA project. Create a .pypirc in your home directory. It should look like this:

```
[distutils]
index-servers =
pypi
pypitest

[pypi]
username=YOUR_USERNAME
password=YOUR_PASSWORD
```

Then run `chmod 600 ./.pypirc` so that only you can read/write the file.

22.2 Part 2: Release Steps

1. Ensure that you're in the top-level `eva` directory.
2. Ensure that your branch is in sync with the `master` branch:

```
$ git pull origin master
```

3. Add a new entry in the Changelog for the release.

```
##  [0.0.6]
### [Breaking Changes]
### [Added]
### [Changed]
### [Deprecated]
### [Removed]
```

Make sure `CHANGELOG.md` is up to date for the release: compare against PRs merged since the last release.

4. Update version to, e.g. `0.0.6` (remove the `+dev` label) in `eva/version.py`.
5. Commit these changes and create a PR:

```
git checkout -b release-v0.0.6
git add . -u
git commit -m "[RELEASE]: v0.0.6"
git push --set-upstream origin release-v0.0.6
```

6. Once the PR is approved, merge it and pull master locally.
7. Tag the release:

```
git tag -a v0.0.6 -m "v0.0.6 release"
git push origin v0.0.6
```

8. Build the source and wheel distributions:

```
rm -rf dist build  # clean old builds & distributions
python3 setup.py sdist  # create a source distribution
python3 setup.py bdist_wheel  # create a universal wheel
```

9. Check that everything looks correct by installing the wheel locally and checking the version:

```
python3 -m venv test_evadb  # create a virtualenv for testing
source test_evadb/bin/activate  # activate virtualenv
python3 -m pip install dist/evadb-0.9.1-py3-none-any.whl
python3 -c "import eva; print(eva.__version__)"
```

10. Publish to PyPI

```
pip install twine  # if not installed
twine upload dist/* -r pypi
```

11. A PR is automatically submitted (this will take a few hours) on [*conda-forge/eva-feedstock*](https://github.com/conda-forge/eva-feedstock) to update the version. * A maintainer needs to accept and merge those changes.
12. Create a new release on Github. * Input the recently-created Tag Version: `v0.0.6` * Copy the release notes in `CHANGELOG.md` to the GitHub tag. * Attach the resulting binaries in (`dist/evadb-x.x.x.*`) to the release. * Publish the release.
13. Update version to, e.g. `0.9.1+dev` in `eva/version.py`.
14. Add a new changelog entry for the unreleased version in *CHANGELOG.md*:

```
##  [Unreleased]
### [Breaking Changes]
### [Added]
### [Changed]
### [Deprecated]
### [Removed]
```

15. Commit these changes and create a PR:

```
git checkout -b bump-v0.9.1+dev
git add . -u
git commit -m "[BUMP]: v0.9.1+dev"
git push --set-upstream origin bump-v0.9.1+dev
```

16. Add the new tag to the EVA project on ReadTheDocs,

- Trigger a build for main to pull new tags.
- Go to the `Versions` tab, and `Activate` the new tag.
- Go to Admin/Advanced to set this tag as the new default version.
- **In `Overview`, make sure a build is triggered:**
  - For the tag `v0.9.1`
  - For `latest`

Credits: Snorkel

CHAPTER

TWENTYTHREE

PACKAGING

This section describes practices to follow when packaging your own models or datasets to be used along with EVA.

23.1 Models

Please follow the following steps to package models:

- Create a folder with a descriptive name. This folder name will be used by the UDF that is invoking your model.
- **Place all files used by the UDF inside this folder. These are typically:**
  - Model weights (The .pt files that contain the actual weights)
  - Model architectures (The .pt files that contain model architecture information)
  - Label files (Extra files that are used in the process of model inference for outputting labels.)
  - Other config files (Any other config files required for model inference)
- Zip this folder.
- Upload the zipped folder to this link inside the models folder.

23.2 Datasets

Please follow the following steps to package datasets:

- Create a folder for your dataset and give it a descriptive name.
- This dataset folder should contain 2 sub-folders named 'info' and 'videos'. For each video entry in the videos folder, there should be a corresponding CSV file in the info folder with the same name. The structure should look like:

```
anipi@blade:~/codes/eva$ tree data/datasets/bddtest/
data/datasets/bddtest/
├── info
│   ├── 00a04f658c891f94.csv
│   ├── 00a04f65af2ab984.csv
│   ├── 00a0f0083c67908e.csv
│   ├── 00a0f008a315437f.csv
│   ├── 00a2e3ca5c856cde.csv
│   ├── 00a2e3ca62992459.csv
│   ├── 00a2f5b6d4217a96.csv
│   ├── 00a395fed60c0b47.csv
│   ├── 00a820ef2b98dcf5.csv
│   └── 00a9cd6bb39be004.csv
└── videos
    ├── 00a04f658c891f94.mp4
    ├── 00a04f65af2ab984.mp4
    ├── 00a0f0083c67908e.mp4
    ├── 00a0f008a315437f.mp4
    ├── 00a2e3ca5c856cde.mp4
    ├── 00a2e3ca62992459.mp4
    ├── 00a2f5b6d4217a96.mp4
    ├── 00a395fed60c0b47.mp4
    ├── 00a820ef2b98dcf5.mp4
    └── 00a9cd6bb39be004.mp4
```

- The videos folder should contain the raw videos in a standard format like mp4 or mov.
- The info folder should contain the meta information corresponding to each video in CSV format. Each row of this CSV file should correspond to 1 unique object in a given frame. Please make sure the columns in your CSV file exactly match to these names. Here is a snapshot of a sample CSV file:

	A	B	C	D	E	F	G
1	id	frame_id	video_id	dataset_name	label	bbox	object_id
2	5699	0	3	bddtest	car	798.6408024113638, 240.2771362586605, 1052.2802666724026, 400.0394580512265	65349
3	5700	0	3	bddtest	car	739.5381062355658, 261.8937644341801, 845.1270207852194, 360	65350
4	5701	0	3	bddtest	car	707.9445727482679, 251.91685912240186, 807.7136258660508, 339.2147806004619	65351
5	5702	0	3	bddtest	car	478.717964316695, 272.0163071719262, 520.0833211152326, 298.941818301956	65352
6	5703	0	3	bddtest	car	688.5975996216332, 253.2197568376064, 731.6349495979853, 324.2814277287924	65353
7	5704	0	3	bddtest	car	645.4521625163827, 267.5229357798165, 675.64875491481, 297.71952817824376	65354

The columns represent the following:

  - id - (Integer) Auto incrementing index that is unique across all files (Since the CSV files are written to the same meta table, we want it to be unique across all files)
  - frame_id - (Integer) id of the frame this row corresponds to.
  - video_id - (Integer) id of the video this file corresponds to.
  - dataset_name - (String) Name of the dataset (should match the folder name)
  - label - (String) label of the object this row corresponds to.
  - bbox - (String) comma separated float values representing x1, y1, x2, y2 (top left and bottom right) coordinates of the bounding box

    - object_id - (Integer) unique id for the object corresponding to this row.

- Zip this folder.
- Upload the zipped folder to this link inside the datasets folder.

Note: In the future, will provide utility scripts along with EVA to download models and datasets easily and place them in the appropriate locations.

CHAPTER

TWENTYFOUR

VERSIONS

- stable version
- v0.2.1
- v0.2.0