# ChatGPT

The provided code is a Flask application that integrates multiple functionalities for video processing, vehicle tracking, and traffic analysis using machine learning models. Here's a detailed breakdown of each part of the code and its logic:

## 1. Flask App Setup:

```python
app = Flask(__name__)
app.config['SECRET_KEY'] = 'vedant363'
```

- **Flask** is initialized with `__name__`. The `SECRET_KEY` is used for securely signing session cookies.

## 2. Global Variables:

```python
global_sheet = None
traffic_analysis_data = {}
```

- `global_sheet` stores the Google Sheets connection.
- `traffic_analysis_data` stores the traffic analysis result.

## 3. Google Sheets API Setup:

```python
SERVICE_ACCOUNT_FILE = 'festive-utility-444415-f5-1d1bf70d3c99.json'
SCOPES = ['https://www.googleapis.com/auth/spreadsheets.readonly',
'https://www.googleapis.com/auth/spreadsheets']
creds = Credentials.from_service_account_file(SERVICE_ACCOUNT_FILE, scopes=SCOPES)
```

- **Google Sheets API** credentials are initialized using a service account. The `SCOPES` define the access permissions for reading and writing to Google Sheets.

## 4. Login Form Setup:

```python
class LoginForm(FlaskForm):
    username = StringField('Username', validators=[DataRequired(), Length(min=4, max=20)])
    password = PasswordField('Password', validators=[DataRequired(), Length(min=6)])
    submit = SubmitField('Login')
```

- A **FlaskForm** for handling login. It validates that the username and password are of the correct length and non-empty.

## 5. Google Sheets Helper Functions:

- **Initializing Google Sheets**:

```python
def initialize_google_sheets(sheet_name):
    client = gspread.service_account(filename=SERVICE_ACCOUNT_FILE)
    sheet = client.open(sheet_name).sheet1
    return sheet
```

- Connects to Google Sheets using the `gspread` library, loads the first sheet of the provided `sheet_name`.
- **Logging Data to Sheets**:

```python
def log_to_google_sheets(timestamp, x1, y1, x2, y2, class_name, confidence, track_id):
    if global_sheet is None:
        global_sheet = initialize_google_sheets('vehicle-detection')

    if not global_sheet.row_values(1):
        headers = ['Timestamp', 'X1', 'Y1', 'X2', 'Y2', 'Width', 'Height', 'Class Name', 'Confidence', 'Track ID']
        global_sheet.insert_row(headers, 1)

    width = x2 - x1
    height = y2 - y1
```

```
    row = [timestamp, x1, y1, x2, y2, width, height, class_name, confidence,
track_id]
    global_sheet.append_row(row)
```

- Logs the vehicle detection data to Google Sheets, including the timestamp, bounding box coordinates, confidence score, and class name. If it's the first entry, headers are added.

## 6. Bounding Box Intersection Over Union (IoU):

python

```python
def box_iou(box1, box2):
    poly1 = shapely_box(box1[0], box1[1], box1[2], box1[3])
    poly2 = shapely_box(box2[0], box2[1], box2[2], box2[3])
    iou = poly1.intersection(poly2).area / poly1.union(poly2).area
    return iou
```

- This function calculates the **Intersection over Union (IoU)** for two bounding boxes using the **Shapely** library. IoU is used to determine how much two boxes overlap.

## 7. Vehicle Tracker Class:

python

```python
class VehicleTracker:
    def __init__(self, max_age=30):
        self.vehicles = {}
        self.max_age = max_age
```

- The `VehicleTracker` class tracks vehicles by storing positions and last seen time. The `max_age` controls how long a vehicle will be tracked before being removed.
- **Updating the Vehicle Tracker**:

python

```python
def update(self, detections):
    current_ids = set()
    for detection in detections:
```

```
        track_id = detection[6]
        if track_id != -1:
            current_ids.add(track_id)
            if track_id not in self.vehicles:
                self.vehicles[track_id] = {'positions': deque(maxlen=30),
'last_seen': 0, 'type': detection[5]}
            self.vehicles[track_id]['positions'].append(detection[:4])
            self.vehicles[track_id]['last_seen'] = 0
    for track_id in list(self.vehicles.keys()):
        if track_id not in current_ids:
            self.vehicles[track_id]['last_seen'] += 1
            if self.vehicles[track_id]['last_seen'] > self.max_age:
                del self.vehicles[track_id]
```

- Tracks vehicle positions for up to 30 frames. If a vehicle is not seen for `max_age` frames, it is removed.
- **Getting Vehicle Speed**:

python

```
def get_vehicle_speed(self, track_id, pixels_per_meter):
    if track_id in self.vehicles and len(self.vehicles[track_id]['positions']) > 1:
        start = self.vehicles[track_id]['positions'][0]
        end = self.vehicles[track_id]['positions'][-1]
        distance_in_pixels = np.sqrt((end[0] - start[0]) ** 2 + (end[1] - start[1])
** 2)
        distance_in_meters = distance_in_pixels / pixels_per_meter
        time_in_seconds = len(self.vehicles[track_id]['positions']) / 30
        speed_meters_per_second = distance_in_meters / time_in_seconds if
time_in_seconds > 0 else 0
        speed_kmh = speed_meters_per_second * 3.6
        return speed_kmh
    return 0
```

- This function calculates the **vehicle's speed** in km/h based on the distance traveled in pixels, converting it into meters and then using the frame count to calculate speed.

## 8. Traffic Analysis:

python

```
class TrafficAnalyzer:
    def __init__(self, road_area, heavy_vehicle_threshold=0.3):
        self.road_area = road_area
        self.heavy_vehicle_threshold = heavy_vehicle_threshold
        self.vehicle_tracker = VehicleTracker()
```

- `TrafficAnalyzer` tracks overall traffic conditions, including the number of vehicles and heavy vehicles, as well as average speed.
- **Analyzing Traffic**:

python

```
def analyze_traffic(self, detections):
    self.vehicle_tracker.update(detections)

    vehicle_count = len(self.vehicle_tracker.vehicles)
    heavy_vehicle_count = sum(1 for v in self.vehicle_tracker.vehicles.values() if
v['type'] in [5, 7, 80])

    speeds = [self.vehicle_tracker.get_vehicle_speed(id, 100) for id in
self.vehicle_tracker.vehicles]
    avg_speed = np.mean(speeds) if speeds else 0

    is_traffic_jam = avg_speed < 5 and vehicle_count > 10
    too_many_heavy_vehicles = heavy_vehicle_count > 30

    estimated_clearance_time = self.estimate_clearance_time(vehicle_count,
avg_speed)

    traffic_light_decision = self.decide_traffic_light(is_traffic_jam,
too_many_heavy_vehicles, avg_speed)

    return {
        'vehicle_count': vehicle_count,
        'avg_speed': avg_speed,
        'is_traffic_jam': is_traffic_jam,
        'too_many_heavy_vehicles': too_many_heavy_vehicles,
        'estimated_clearance_time': estimated_clearance_time,
        'traffic_light_decision': traffic_light_decision
    }
```

- The function tracks vehicle count, average speed, and checks for heavy vehicles. It decides whether there's a traffic jam, how many heavy vehicles are present, and estimates traffic clearance time.
- **Estimating Clearance Time:**

```python
def estimate_clearance_time(self, vehicle_count, avg_speed):
    if avg_speed > 0:
        return (vehicle_count * 5) / avg_speed
    return float('inf')
```

- **Traffic Light Decision:**

```python
def decide_traffic_light(self, is_traffic_jam, too_many_heavy_vehicles, avg_speed):
    if is_traffic_jam:
        return 'green', 120
    elif too_many_heavy_vehicles:
        return 'green', 90
    elif avg_speed < 10:
        return 'green', 60
    else:
        return 'red', 30
```

- **Traffic light decisions** are made based on the traffic analysis (e.g., green for traffic jams, or for heavy vehicles).

## 9. Video Stream and Model Inference:

- The rest of the code handles **video capture**, **YOLO model inference**, and **vehicle tracking**. It uses the `YOLO` object detection models to track vehicles and analyze traffic conditions, drawing bounding boxes and labels on the video frames. Each frame is processed to track vehicle movement and log data to Google Sheets.

This Flask app integrates various advanced features such as vehicle tracking, traffic analysis, and Google Sheets integration in a live video stream setting.

Certainly! Let's continue with the explanation for the remaining part of the code.

## Routes for Login:

```python
@app.route('/', methods=['GET', 'POST'])
@app.route('/login', methods=['GET', 'POST'])
def login():
    form = LoginForm()
    if form.validate_on_submit():
        username = form.username.data
        password = form.password.data
        if username == 'admin' and password == 'admin':
            flash('Login successful!')
            return redirect(url_for('video_feed'))
        else:
            flash('Invalid credentials. Please try again.')
    return render_template('login.html', form=form)
```

- **Route Declaration**: This route listens for both `GET` and `POST` requests at the root URL (`/`) and `/login` URL, so that users can access the login page.
- **Form Handling**: It uses the `LoginForm` defined earlier. The `validate_on_submit()` method checks if the form is valid when the user submits it.
- **Authentication**: If the form is valid and the username and password match the hardcoded values (`'admin'`), the user is authenticated, and a success message is flashed. The user is then redirected to the video feed page (`video_feed`).
- **Invalid Login**: If the username or password is incorrect, the code flashes an error message and stays on the login page for the user to try again.

## Flask App Configuration and Running:

```python
if __name__ == '__main__':
    app.run(debug=True)
```

- **Running the Flask App**: The `if __name__ == '__main__':` block ensures that the Flask application runs only when the script is executed directly (not when imported as a

module).

- **App Run**: `app.run(debug=True)` starts the Flask server with debugging enabled, which means that any changes to the code are automatically reflected without restarting the server. It also enables error messages to be shown on the browser in case of any issues.

## Summary of Logic and Flow:

1. **Flask Setup**: A Flask web application is created, with a form to handle user login and a video stream feed for displaying real-time traffic analysis.
2. **Login System**: The user can log in with a username and password (in this case, a simple hardcoded check).
3. **Vehicle Detection and Traffic Analysis**: The application uses the YOLO object detection model to track and analyze vehicles in a video stream, logging vehicle information (e.g., class, coordinates, confidence, etc.) into a Google Sheet.
4. **Google Sheets Integration**: Vehicle detection data (timestamp, vehicle class, position, etc.) is logged to a Google Sheet via the `gspread` library. The data is fetched using Google Sheets API.
5. **Traffic Analysis**: A traffic analyzer estimates the vehicle count, average speed, and makes decisions on traffic light timing based on certain thresholds (traffic jam, heavy vehicles).
6. **Video Feed**: The video feed is continuously processed to detect vehicles, calculate their speed, and display this information on the Flask web interface in real time.
7. **UI and Feedback**: The UI is managed with Flask, displaying live video feed, login form, and flash messages for login status.

Let me know if you need further explanations on any part!