Challenge 2: Evaluating Perception Model in SafeBench

24-784 Special Topics: Trustworthy AI Autonomy

Prof. D. Zhao

Due: Friday Mar 31st, 23:59:59 EST

- Your online version and its timestamp will be used for assessment.
- We will use Gradescope to grade. The link is on the panel of CANVAS. This homework is to be completed **in a group**. Please abide by the ACADEMIC INTEGRITY POLICY on the syllabus.
- Submit your solutions in **.pdf** format to **Challenge-2**. Insert the generated images and plots in the **.pdf**. We will check all of your answers in the report.
- Please familiarize yourself with Python programming, if you have never used it previously. We only accept Python for the coding of all homeworks. To that end, attending Python recitation bootcamp at the beginning of this course is highly recommended.
- We advise you to start with the assignment early. All the submissions are to be done before the respective deadlines. For information about available credits for late days, refer to the syllabus in CANVAS.

1 Introduction

In this challenge assignment C2, you are required to form a team in 3 to tackle a real-world problem in autonomous driving.

Perception system is of crucial importance in autonomous driving, although the prosperity of deep learning has brought huge success in different tasks like image classification, semantic segmentation, and object detection, these deep neural networks are oftern vulnerable against adversarial attack. How to build a robust perception model under different conditions (e.g. different orientations, different illumination conditions, or even different attacks) is still an open questions in the frontier for autonomous driving researchers.

In P1, we've already seen an example where deep neural networks fail to classify the stop sign under PGD attack. In this challenge, we'll step further to see how these attack on the stop sign will function in the object detection task in the physical world.

The goals of this challenge are:

- Learn about the Carla simulator, understand the basic frameworks of SafeBench.
- Implement the evaluation metrics for object detection.
- Evaluate the object detection model under the given textures:
  - A single-stage anchor-based object detectors (YOLOv5),
  - A two-stage object detector with a region proposal network and a recognition network (Faster RCNN).
- Evaluate the impact of the size and position of patch to the detection results.
- Design your customized patch that could successfully attack object detection model.

2 Resources

2.1 CARLA Simulator

CARLA is an open-source autonomous driving simulator. It was built from scratch to serve as a modular and flexible API to address a range of tasks involved in the problem of autonomous driving. One of the main goals of CARLA is to help democratize autonomous driving R&D, serving as a tool that can be easily accessed and customized by users. To do so, the simulator has to meet the requirements of different use cases within the general problem of driving (e.g. learning driving policies, training perception algorithms, etc.). CARLA is grounded on Unreal Engine to run the simulation and uses the OpenDRIVE standard (1.4 as today) to define roads and urban settings. Control over the simulation is granted through an API handled in Python and C++ that is constantly growing as the project does.

To get more detailed introduction to CARLA, you are welcome to check the documents of CARLA_0.9.13, which is the exact CARLA version we will use for this challenge.

2.2 SafeBench

SafeBench is a benchmarking platform based on CARLA simulator for evaluating safety and security of autonomous vehicles. The info flowchart of different modules is shown in Fig. 1.

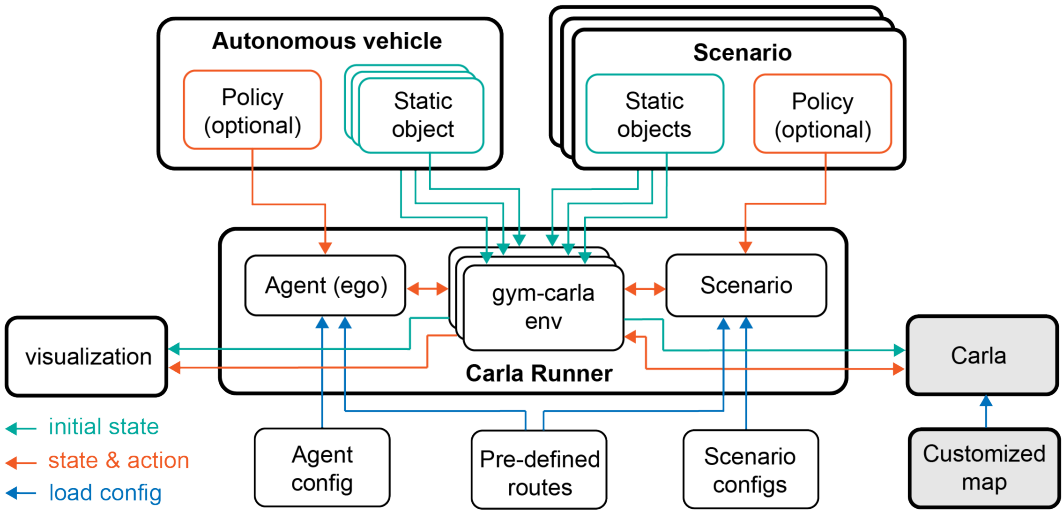

Figure 1: Pipeline of SafeBench Platform. The center of the environments is a CARLA environments with Open AI gym-styled interface. There are two major components that communicate with the environments: (i) Agent module, which controls the perception or planning of ego vehicles; (ii) Scenario module, which controls the static or dynamic environments that interacts with the agents.

This benchmark supports three types of running:

- **Train Agent**: train policy model of autonomous vehicle with pre-defined scenarios.
- **Train Scenario**: train policy model of scenario participants against pre-trained autonomous vehicles.
- **Evaluation**: fix both autonomous vehicle and scenario to do evaluation.

For challenge C2, we'll ONLY use **Evaluation** mode to produce all the results you need in the report. The detailed configuration of SafeBench on a remote server can be found in here. We'll make a more detailed tutorial session to cover all the related operations you need to start working on this platform.

2.3 AWS Platform

We recommend you to finish this challenge on AWS servers. Here are some necessary components that you'll find useful in setting up the platform.

- **EC2 Instance**: We recommend you to select g5.xlarge for this and the following challenge. You may encounter the error about your vCPU limit, which requires you to apply for a higher limits by specifically introduce your use case. Here is a tutorial of how you could calculate the limits and increase it (to 4 vCPU for g5.xlarge).
- **Remote Desktop**: We use **TurboVNC** for the remote server. You can download the .deb package from here. Select the latest version with "_amd64" suffix.

3 AWS Instance Instruction

Starting from this challenge, you will be using CARLA simulator to conduct real-world perception and RL problems! There will be four steps in successfully setting up the necessary environments on an **AWS server**. Please refer to this instruction, and follow our tutorial video (which you could find on canvas file later) in launching a server from scratch.

Step 1: AWS EC2 Instance application and launching

Step 1.1: Apply for GPU usage: apply in US East (North Virginia).

Step 1.2: Apply for vCPU (specify the reason) You can follow this tutorial to accomplish this application. In your EC2 management page, click 'Limits', search for the current vCPU limits on all G instances and select them, then click 'request limit increase' on the right upper side, which will direct you to the support center of AWS.

In the support center page, select Limit type as EC2 instances. In the 'requests' section, select region as US East (Northern Virginia), and Primary Instance Type as All G instances. Select the new limit value to 4. In the case description section, describe the potential use

case in a detailed way.

An example template for the case description:

In the course called Trustworthy AI and Autonomy at CMU, we aim to evaluate perception models for autonomous driving in the CARLA simulator, which will conduct object detection by different models, e.g. YOLO-v5, Faster-RCNN, etc. As we build up a training pipeline via CARLA gym APIs, which needs parallel data collection and model finetuning based on the sampled data. Specifically, we're interested in the robustness of the model, which may require computing resources for the adversary. The simulator data rollouts as well as model training could be costly for CPUs, thus we hope to increase the quota limit for vCPU, and we've fully understood the cost of our target instance. Our challenge is attached in: https://github.com/trust-ai/SafeBench.

Step 1.3: Select the instances you need

- **OS Image**: ubuntu 22.04
- **Instance type**: g5.xlarge
- **Key pair** for login: generate a `<your_key_name>.pem` file and download it to your local device **(keep it!)**

If step 1.1 or 1.2 is not finished, even if you select certain instance, the launch would be failed. Therefore, please make sure that both of your applications in 1.1 and 1.2 are approved.

Step 2: ssh log in and ubuntu system setup

Step 2.1: Log in with your public key file

```
!ssh -i <your_key_name>.pem ubuntu@<your_ip_address>
```

Step 2.2: ssh key generation

```
!ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```

You'll be able to find your ssh key on ∼/.ssh/id_rsa.pub, then you can first use:

```
!ssh -i <your_key.pem> ubuntu@<your_ip_address>
```

to log in to the AWS server, then copy your key at ∼/.ssh/id_rsa.pub (from your laptop) to ∼/.ssh/authorized_keys (on the server) by running the following command on your server:

```
cp ∼/.ssh/authorized_keys ∼/.ssh/authorized_keys_copy
rm ∼/.ssh/authorized_keys
vim ∼/.ssh/authorized_keys
```

Then copy your local ~/.ssh/id_rsa.pub to the new file ~/.ssh/authorized_keys on the server.

Note that you should be able to log in without the .pem keys now:

```
!ssh ubuntu@<your_ip_address>
```

Step 2.3: Install the latest version of NVIDIA driver We'll use GPUs for model inference, while the AWS ubuntu system does not install GPU drivers beforehand. Run the following commands:

```
sudo apt-get install nvidia-driver-525
```

After finishing this command, please stop the instance on the console of AWS, then restart it. Notice that you should check the new ip address, which is changed everytime your start the server.

```
!ssh ubuntu@<your_latest_ip_address>
```

After logging back, you can try nvidia-smi and see whether you get the correct return of your GPU running status.

Step 3: GUI installation and Remote Desktop

Next, we'll install the GUI of ubuntu and the tool of remote desktop, which will be useful in your CARLA running and debugging.

Step 3.1: Install the ubuntu desktop You can run the following command without –fix-broken, if any error occurs, you could add –fix-broken and retry.

```
!sudo apt-get update
!sudo apt-get install (--fix-broken) ubuntu-desktop
```

Step 3.2: Download and Install TurboVNC Download TurboVNC for ubuntu with amd64 system on the server by using:

```
!wget \
https://sourceforge.net/projects/turbovnc/files/3.0.3/turbovnc_3.0.3_amd64.deb/download
!sudo dpkg -i download
```

Also remember to install the TurboVNC **on your local device** (with the corresponding MacOS or Windows package from the https://sourceforge.net/projects/turbovnc/files/)

Then start the TurboVNC on your local device, put `ubuntu@<your_latest_ip_address>` in the 'VNC server' of your local VNC viewer, then click 'connect', you should be able to

see the remote desktop on the server.

Step 4: Configuration of Simulator and Codebase Environments

Congratulations! We're finally close to the end of this environment setup journey.

Step 4.1: Install Anaconda/Miniconda

```
!wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
!bash Miniconda3-latest-Linux-x86_64.sh
```

Then just press enter or inputs 'yes' for the installation of Anaconda.

Step 4.2: Download CARLA to your server:

First, download it to your local device from this google drive link. Next, use the scp command to upload the .tar.gz file to the server:

```
scp <path_to_carla_file.tar.gz> ubuntu@<your_ip_address>:/home/ubuntu/Downloads
```

Then extract the file on the server by running:

```
tar -xzvf /home/ubuintu/Downloads/<carla_file.tar.gz> carla/
```

Remember to Use `vim ~/.bashrc` to edit your bashrc files and insert the following lines (please refer to **installation step 6** in the github repo (starting as):

```
export CARLA_ROOT=path/to/your/carla
....
```

Finally, you should be able to start your Carla simulator:

```
./CarlaUE4.sh -prefernvidia -RenderOffScreen -carla-port=2000
```

Step 4.3: Clone the repo from github and install SafeBench

```
!conda create -n safebench python=3.8
!conda activate safebench
!git clone --branch 24784_s23 https://github.com/trust-ai/SafeBench.git
!cd SafeBench
!pip install -r requirements.txt
!pip install -e .
```

4 Problem Sets

Exercise 1: Different Object Detection Models

For all the following exercises, you'll start to use SafeBench as your experiment platform. The evaluation metrics is average over the **AP@[0.5:0.05:0.95]**, which means you will take the mean Average Precision with 11 IoU threshold as 0.5, 0.55, $\cdots$, 0.95.

You'll use part of your code from Challenge 1 to compute the average precision. Your will need to modify the code in `safebench/util/metric_util.py`. There will be a comment '# TODO TAIAT C2' in two sections: function `_compute_ap` starting from line 122, and `_get_pr_ap` starting from line 136. After you insert the proper code block in computing precision, recall, and AP, you could run your experiment with the following command at your root path of SafeBench:

```
!python scripts/run.py --mode=eval --agent_cfg faster_rcnn.yaml \
--scenario_cfg object_detection_stopsign.yaml --num_scenario 4

!python scripts/run.py --mode=eval --agent_cfg yolo.yaml \
--scenario_cfg object_detection_stopsign.yaml --num_scenario 4
```

Save video: You can save video by adding `--save_video` after the above command, e.g.:

```
!python scripts/run.py --mode=eval --agent_cfg yolo.yaml \
--scenario_cfg object_detection_stopsign.yaml --num_scenario 4 --save_video
```

The output video will by default be stored in the folder `'log/video/'` unless you specify `--output_dir` for the above scripts.

Texture modification: you can change the imported textures in the `'/safebench/scenario/config/object_detection_stopsign.yaml'`, where "texture_dir" is the relevant path of your imported texture to the stop sign objects.

There are three different textures for the stop sign under the directory:
`'safebench/scenario/scenario_data/template_od'`,
named as `'stopsign_*.jpg'`.

Your report should include a table that gives the AP@[0.5:0.05:0.95] results (with mean and standard deviation on 4 different environments) for two object detection models (YOLOv5 and Faster R-CNN) under 3 textures (`stopsign.jpg, stopsign_1.jpg, stopsign_2.jpg`)

Exercise 2: Customized Patch Attack

Now please try to design your own image. Notice that at this stage you don't necessarily need to conduct adversarial attack on the object detection model. Your attached patch should

be smaller than 200x200 in width and height (the center of the stop sign for the 512x512 input texture image is around (310, 310)). Show your detection results in a video of drive-by testing. You just need to report your **AP@[0.5:0.05:0.95]** metrics for both YOLO-v5 and Faster RCNN models. You'll get a full credit if your customized textures achieve an **AP lower than 0.5** for Faster RCNN and **AP lower than 0.8** for YOLO-v5.

Exercise 3: Different Geometric Transformations

In this section, you are required to evaluate your customized patch with different the geometric transformations, e.g. the size, the position, or the rotation of the patch. Illustrate **at least two groups of evaluation against these transformations** of your patch.

Your report for this may include four plots (two on each object detection model) that shows the average of **AP@[0.5:0.05:0.95]** w.r.t. the size, position, or the rotation of the patch. Inlude all the samples you create in your submission files. Fig. 2 and Fig. 3 are some examples that you could consider to demonstrate your results.

Bonus Options

There will be two bonus options for this project. (It's totally fine that you neglect them!)

- Train your patch with some black-box attack baselines.

  You can refer to the torchvision documentation
  and `safebench/agent/object_detection/faster_rcnn.py`, then import other pre-trained models from torchvision in a similar way.

- Train your patch with some black-box attack baselines.

  You can refer to the benchmark of black box attack and
  `safebench/scenario/scenario_policy/adv_patch.py` for more details.

5 Submission

Your final report should include the following contents.

- **Exercise 1**: illustrate and analyze the difference of performance between two models in IoU curves (w.r.t. timesteps) and AP metrics.
- **Exercise 2**: include your customized patch, and how you attach them on the stop sign texture, report the AP@[0.5:0.05:0.95] results under two object detection models.
- **Exercise 3**: give two groups of comparison (in the form table or plot) to evaluate the impact of geometric transformations to AP@[0.5:0.05:0.95] under two object detection models.

Apart from the report, you would upload the videos you record on your customized patch (2 videos under each object detection model) and the results under its different geometric transformations in Exercise 2 and 3 (just upload your google drive links to the videos).

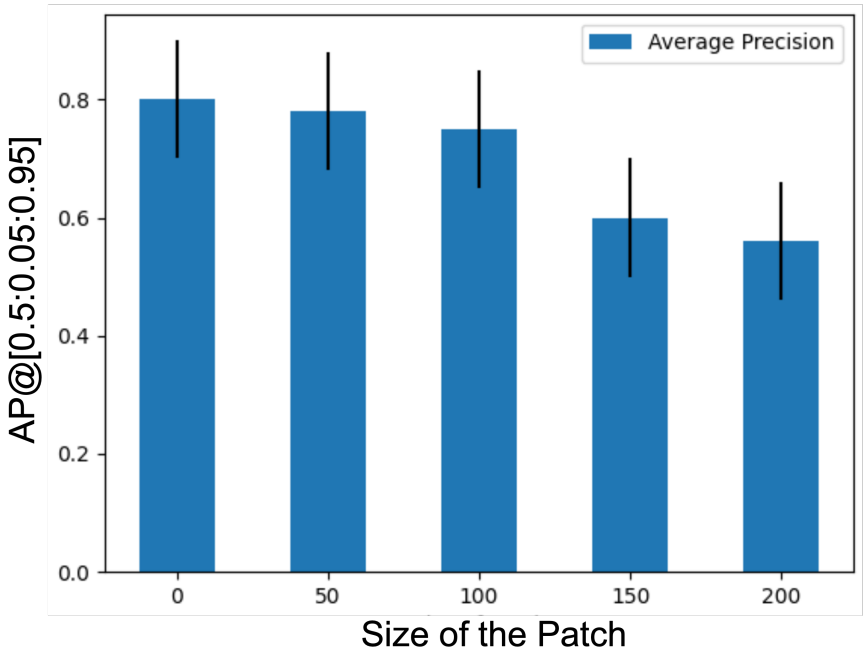

Figure 2: Example for impact of the size of patch to the AP@0.5 metrics.

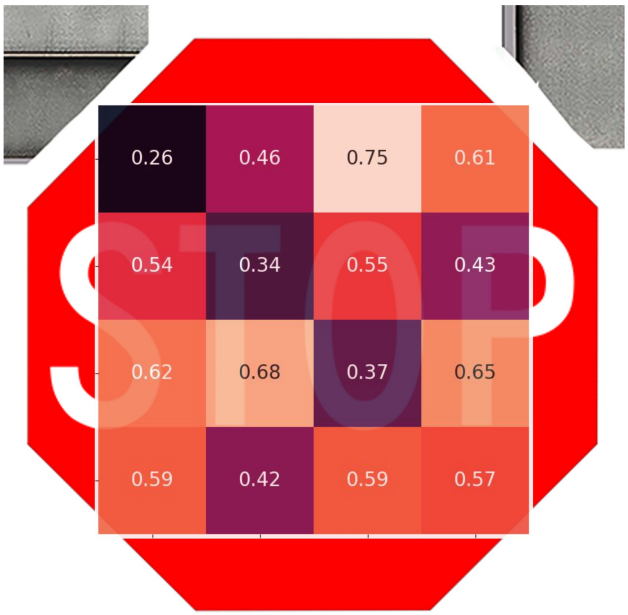

Figure 3: Example for heatmap of the position's impact of to the AP@0.5 metrics. You can select the grid you search over, and please mention your size of patch in your report.

6 Reference

1. AWS setup, https://www.youtube.com/watch?v=Yn-w3sfJil4.
2. SafeBench: https://github.com/trust-ai/SafeBench/tree/24784_s23
3. Object Detection: https://www.jeremyjordan.me/object-detection-one-stage/
4. TorchVision models, https://pytorch.org/vision/0.8/models.html
5. YOLOv5: https://github.com/ultralytics/yolov5
6. Faster R-CNN: https://blog.paperspace.com/faster-r-cnn-explained-object-detection/