Lab 4 Report

Member Names: Rahul Singhal, Anand Todkar, Constantin Kappel
NetID: rahuls11, atodkar2, normank2
Late days used: 0
Video Link: Drive Video Link
Repo Link: UIUC Engr Gitlab
Contribution: All members: Anand, Constantin, and Rahul contributed equally to this lab.

Section 1: Build the Cloud

Please watch the video.

Section 2: Data Inference

An effective design approach for managing emissions data in a fleet of vehicles is to use a long-lived AWS Lambda function that maintains two global dictionaries: one for the maximum CO2 value for each vehicle and another for the current timestep. After receiving a message, the Lambda function publishes the maximum CO2 value to a topic called **emissions/vehN**, where N represents the vehicle number. This topic-based approach ensures that each device in the fleet is allocated a subscription only to its corresponding vehicle. There is one subscription going in each direction between device and Lambda function, plus one additional subscription from Lambda to IoT Cloud for capturing IoT Analytics data of the processed emissions for post processing and visualization.

Section 3: Analytics

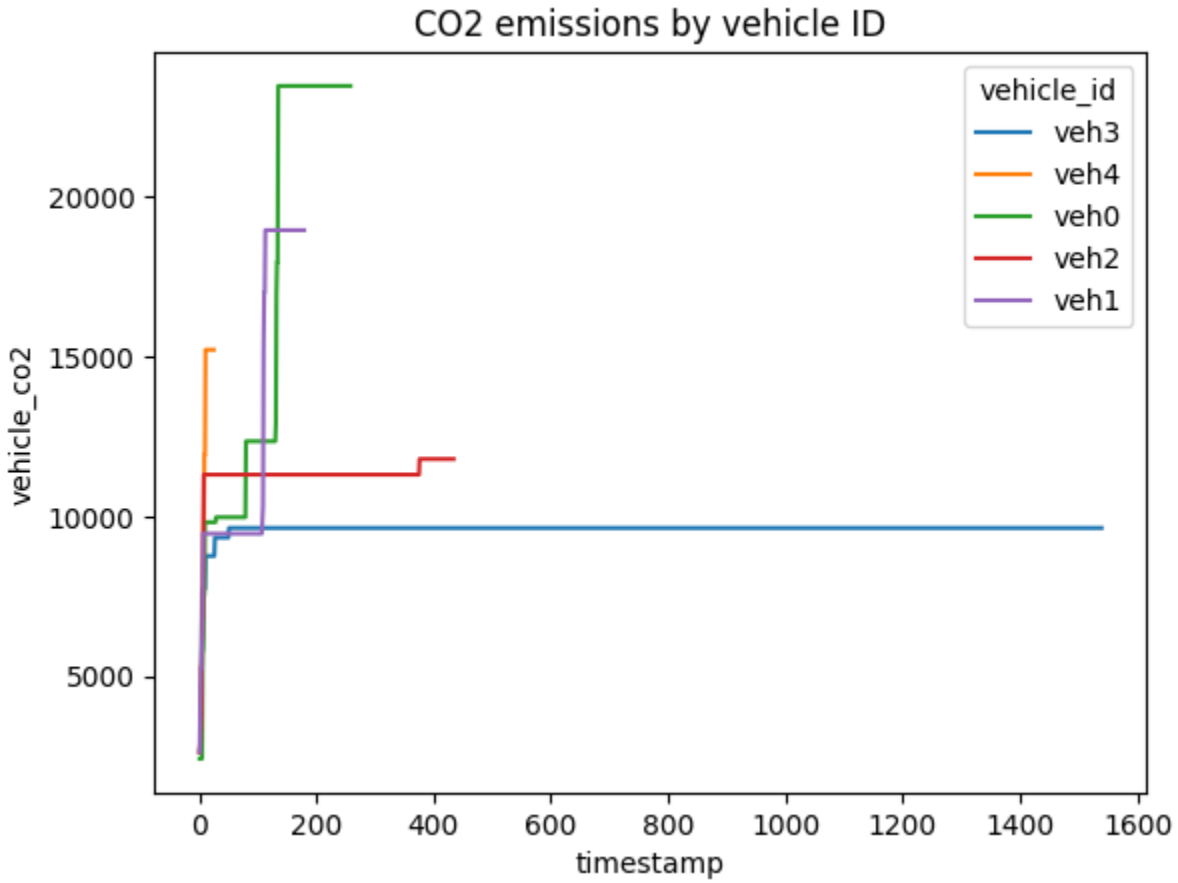
We used the greengrass core and lambda functions set up in part 2 to funnel data in AWS IoT Analytics.

To this end we configured the following in AWS IoT Analytics:

1) A channel, which subscribes to the related topic “emissions/#” to funnel the data into the pipeline
2) A pipeline which feeds it into a data store.
3) A data store which holds the data
4) A dataset which holds the data in a table.

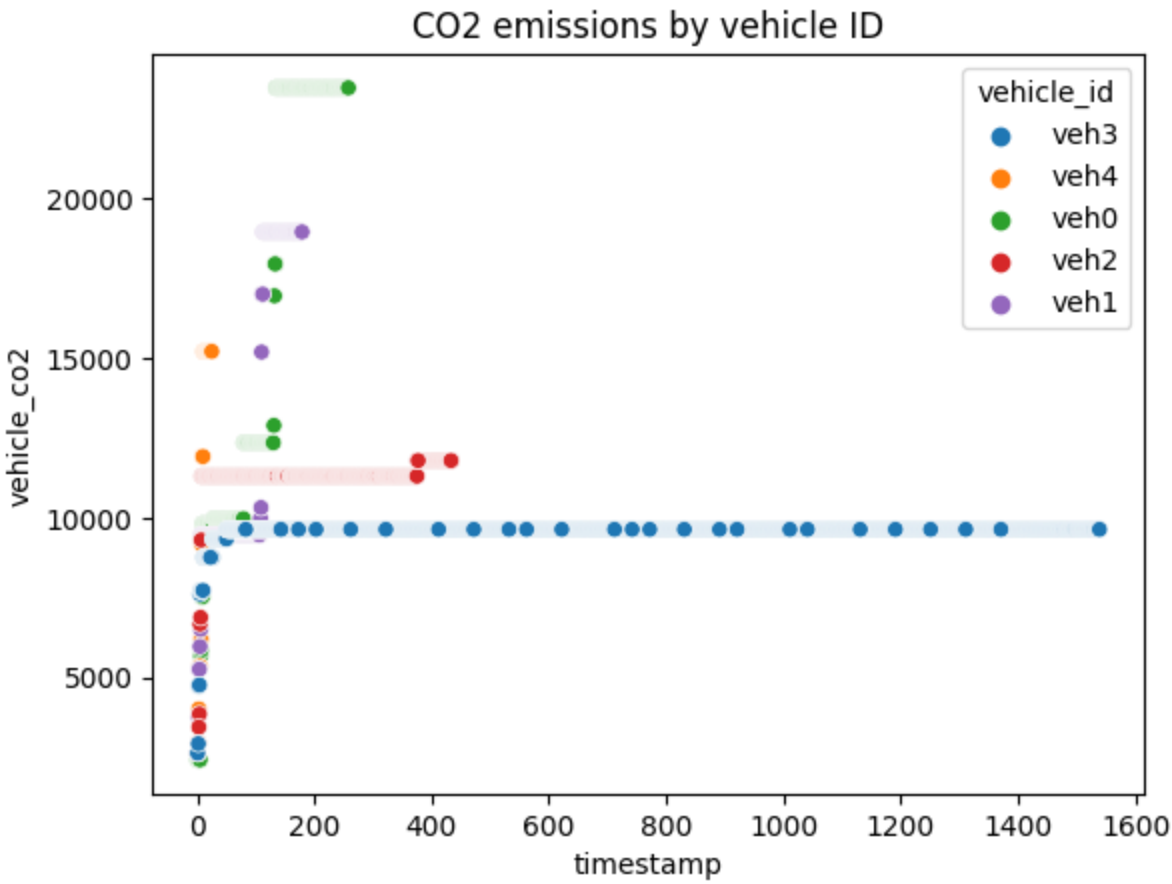
5) A Jupyter Notebook on Sagemaker which lets us load and view the data

For plotting our data the following ways might make sense. A line plot shows us the evolution of maximal carbon emissions per vehicle over time. We chose arbitrary time units as in the CSV files.

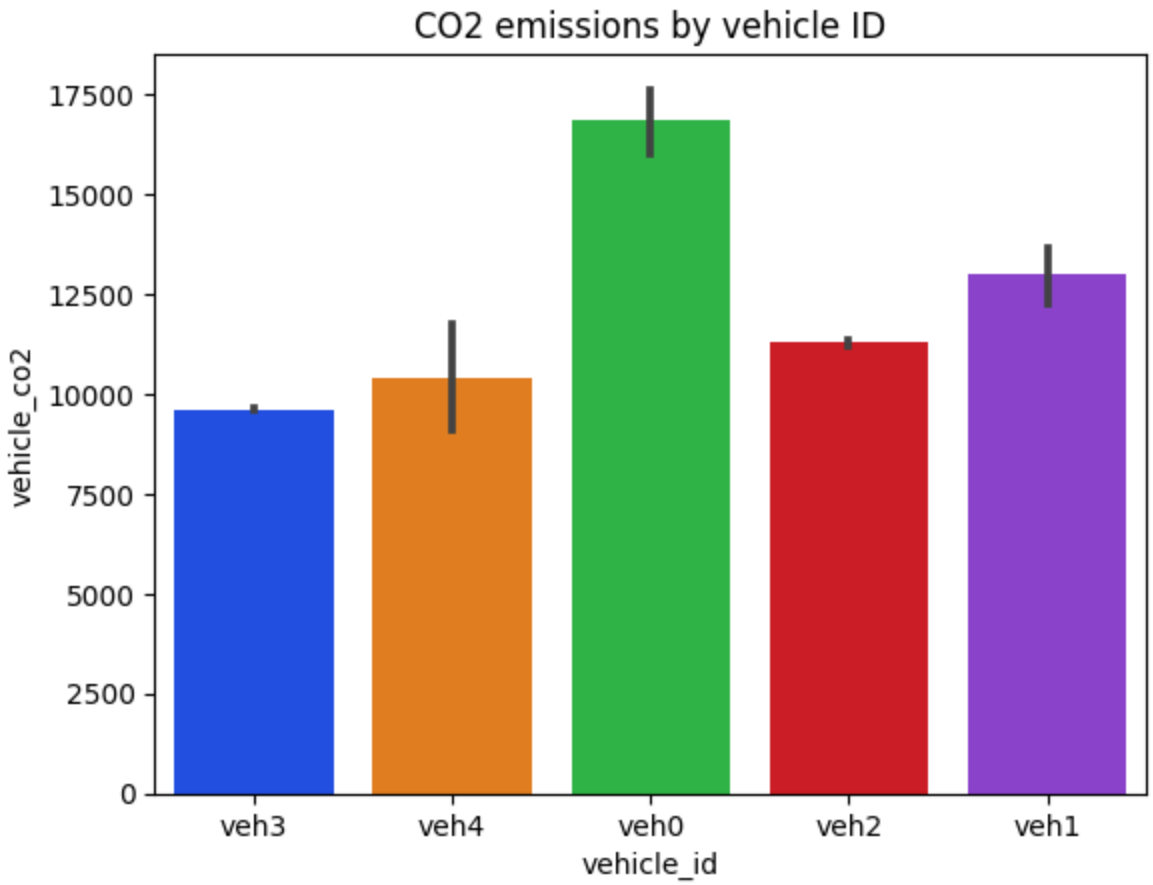


We could interpret this as in terms of which vehicle reaches its maximum CO2 emissions first. Note, this would not, however, be an indication of which vehicle produces the most CO2, because we are only looking at the maximum value. In other words, e.g. vehicle 0 has the highest maximum emission, but it might have spiked and then gone down again. We can tell that the measurements were the longest (as in number of measurements or samples) for vehicle 3 and shortest for vehicle 4.

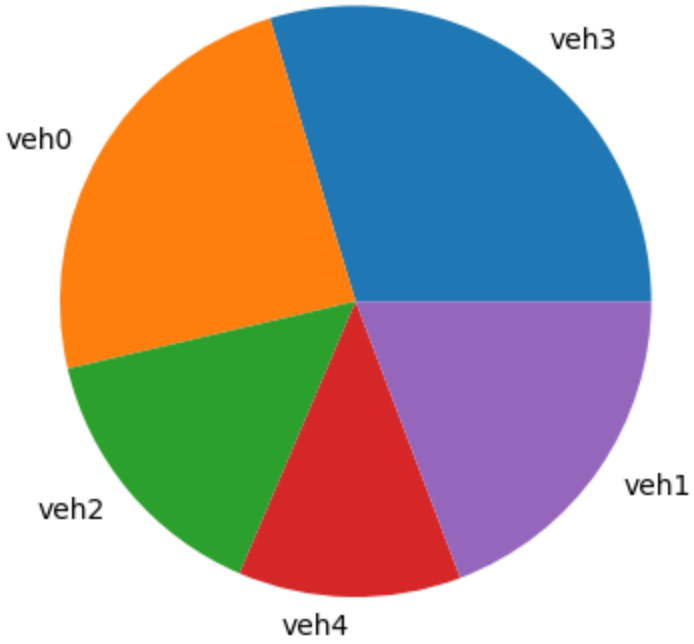
A variant of this might be the scatter plot. In this example the line plot would be easier to read, though



A bar plot gives us a better overview of which car contributes the most to maximal carbon emissions. Note, however, in order to get absolute values we would have to use the integral (cumulative sum) over all data, not the maximum which our lambda function reported.



In the pie chart below we get a good overview of the relative contributions of each vehicle's maximum CO2 emissions.

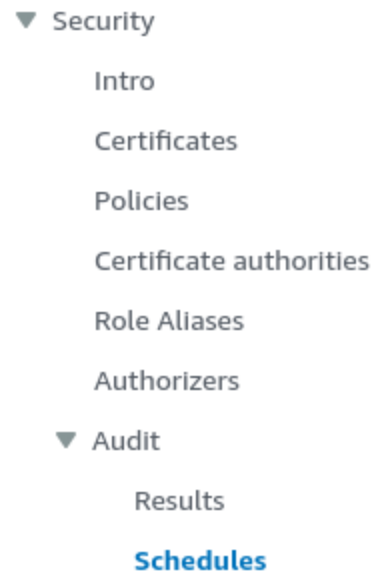


Section 4: AWS IoT Device Defender

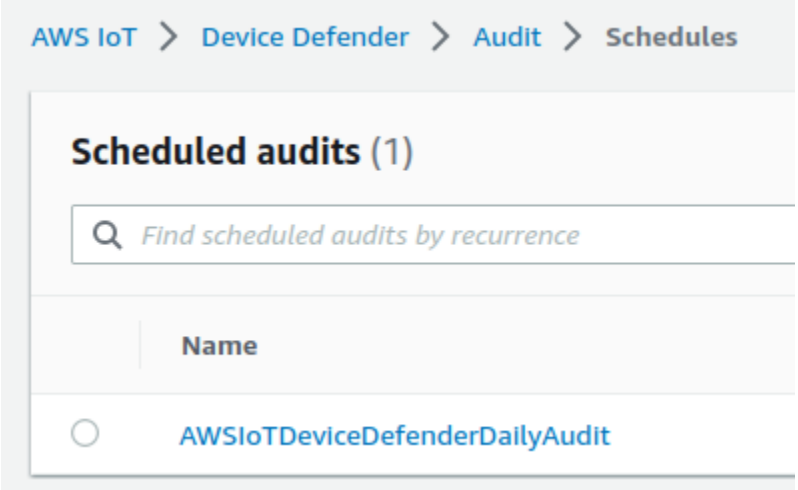
In this section we deal with potential security issues in two ways:

1) Through a security audit
2) Through anomaly detection

The former starts with setting up a security audit in



We schedule a daily security audit



AWSIotDefender allows us to select a number of areas which get audited, such as key expiries or security policies:

Scheduled audit ARN uniquely identifies this scheduled audit.

`arn:aws:iot:us-east-1:662440407589:scheduledaudit/AWSIoTDeviceDefenderDailyAudit`

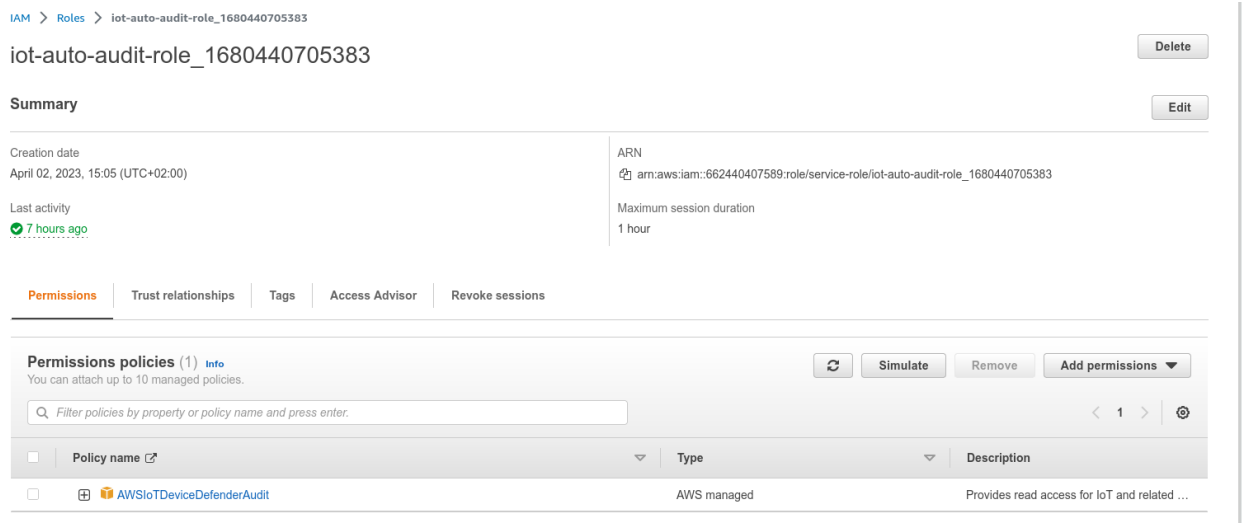
Details

Recurrence: Daily

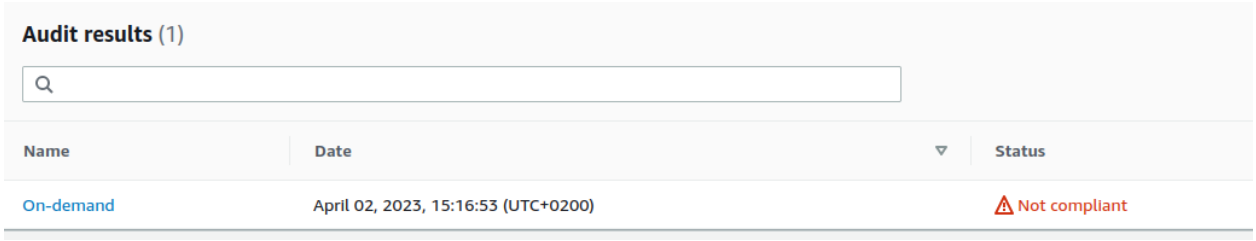
Checks

Check name	Severity	Resource type
Authenticated Cognito role overly permissive Info	Critical	Cognito pool
CA certificate expiring Info	Medium	CA certificate
CA certificate key quality Info	Critical	CA certificate
Conflicting MQTT client IDs Info	High	Client ID
Device certificate expiring Info	Medium	Device certificate
Device certificate key quality Info	Critical	Device certificate
Device certificate shared Info	Critical	Device certificate
Intermediate CA revoked for active device certificates Info	Critical	Issuer certificate
IoT policy overly permissive Info	Critical	Policy
IoT policy potentially misconfigured Info	Medium	Policy
Role alias allows access to unused services Info	Medium	Role alias
Role alias overly permissive Info	Critical	Role alias
Logging disabled Info	Low	Account settings
CA certificate revoked but device certificates still active Info	Critical	CA certificate
Revoked device certificate still active Info	Medium	Device certificate
Unauthenticated Cognito role overly permissive Info	Critical	Cognito pool

On a side node: The audit we set up creates an IAM role which can assume the rights to perform this.

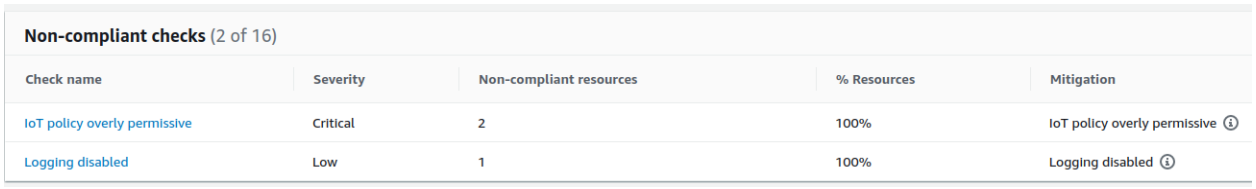


Once run the audit results are visible and give a top level summary:

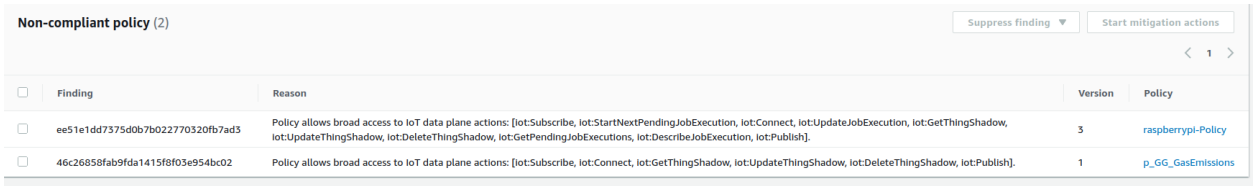


Not Compliant is demonstration of the Audit working.

Drilling down reveals the most critical issues are around security policies being set too permissively:

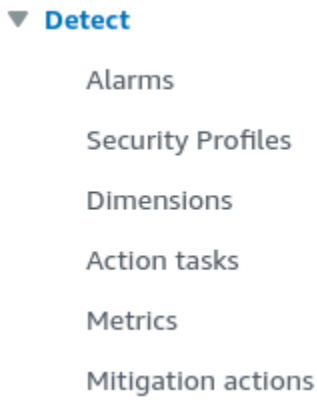


We can see which policies are affected and can take action:



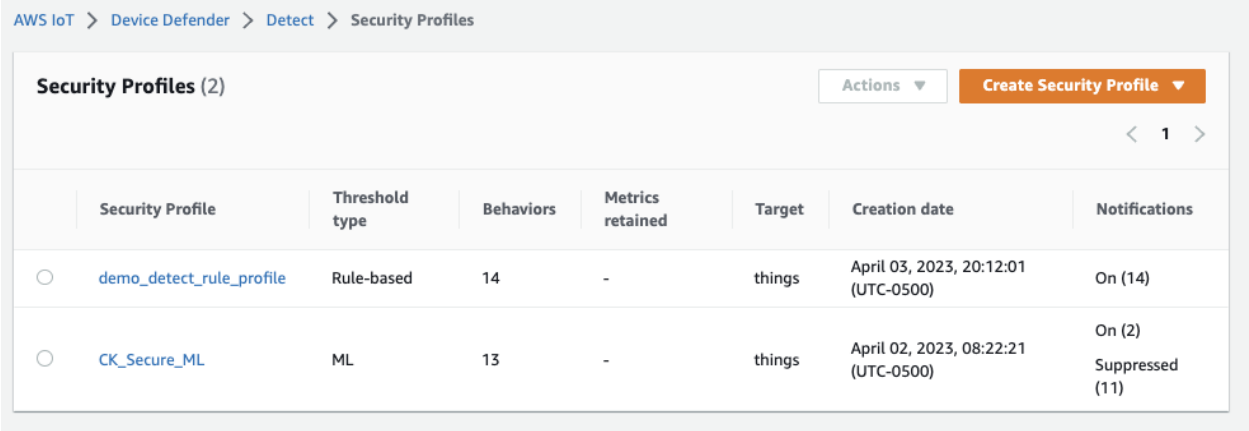
This audit is a more static analysis of the current status quo (which, however, can change every day as devices and “things” are added or removed).

An active check is monitoring of suspicious activity by anomaly detection.
This is found in Security/Detect



We can set a security profile and get to choose to use either machine learning or rule-based strategies for anomaly detection. If the expected behavior is well understood, then a rule-based

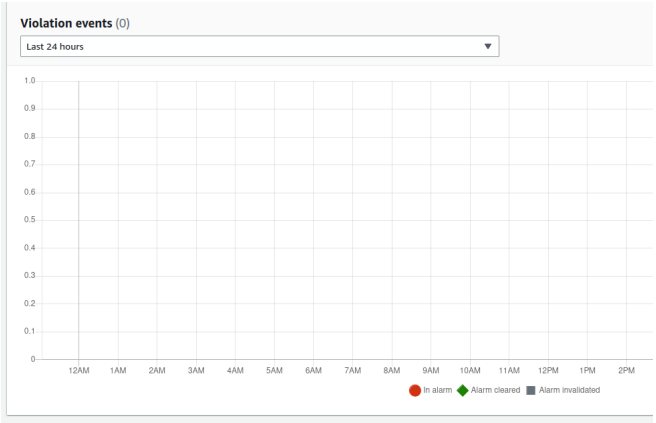
system is sufficient, otherwise an ML model can be used for unknown behaviors or changing behavior.



For each profile we get a history (in our case it is empty, there has been no activity yet, because ml models require 14 days and minimum of 25k data points).

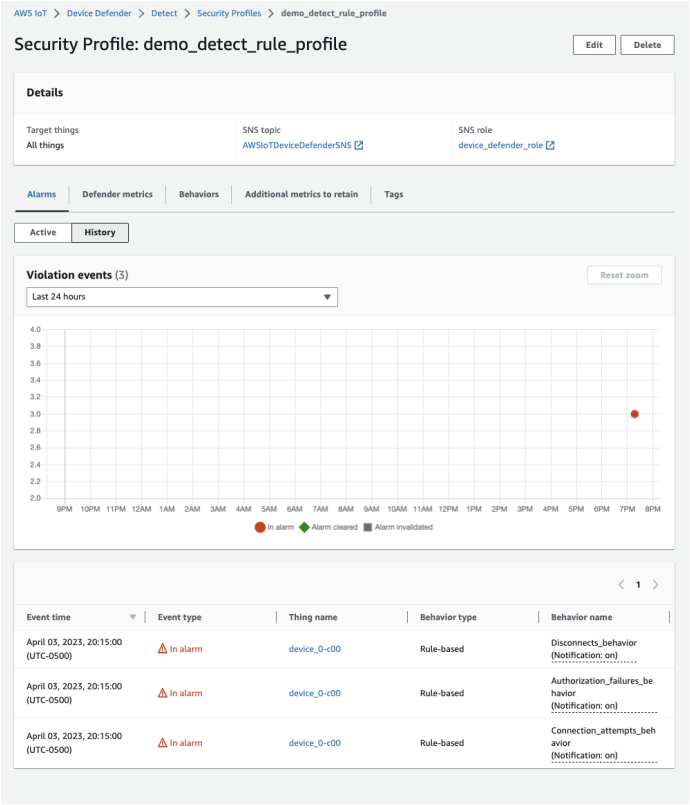
Q. How does AWS IoT Device Defender ML Detect model training work?

AWS IoT Device Defender uses machine-learning models to monitor and identify anomalous datapoints for device behavior metrics in ML Detect. While AWS IoT Device Defender is building its initial ML model for your devices, it requires 14 days and a minimum of 25,000 metric datapoints per metric to generate the model. Afterwards, it updates the model every day as long as the minimum 25,000 metric datapoints per metric are met. If the minimum datapoint requirement is not met, AWS IoT Device Defender will attempt to update the model on the next day. It will retry daily for 30 days before discontinuing the model updating.



And we can set metrics and thresholds which can trigger alarms or predefined mitigation actions.

Rule-based systems can more easily be tested by setting low thresholds.

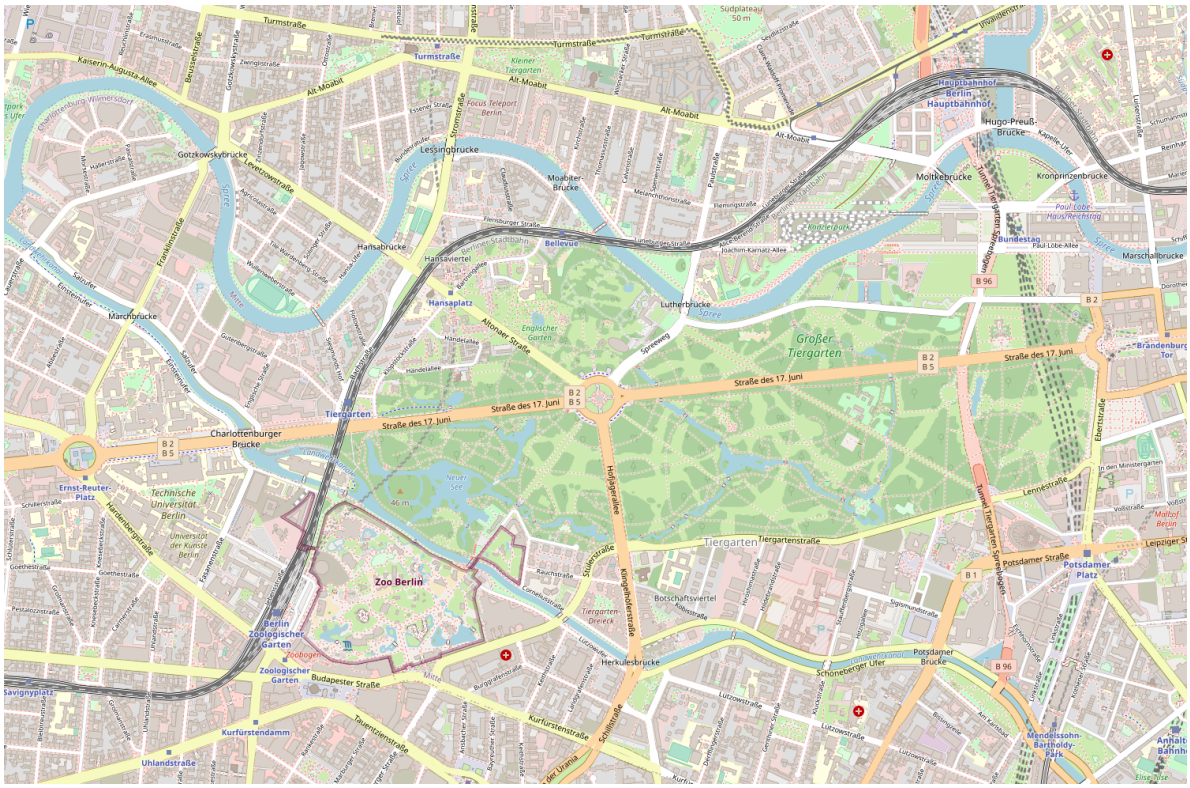


Appendix 1: Experiment with other networks

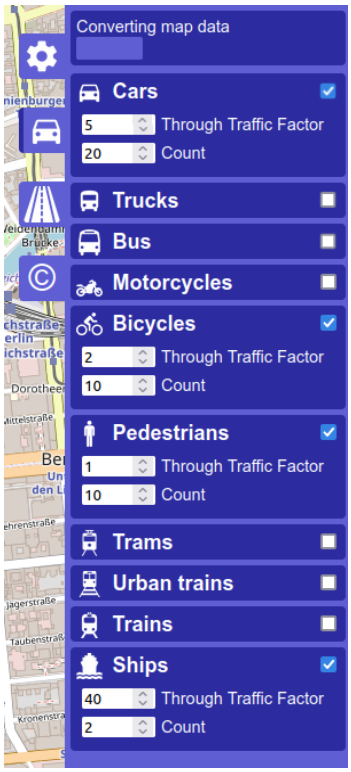
This part is about simulating traffic on a map using the actual OpenStreetMap topology. We chose a part of Berlin/Germany around “Tiergarten” (the Zoo).

We did simulations on two different sizes of this area. The “small one”:

And the large size comprises:

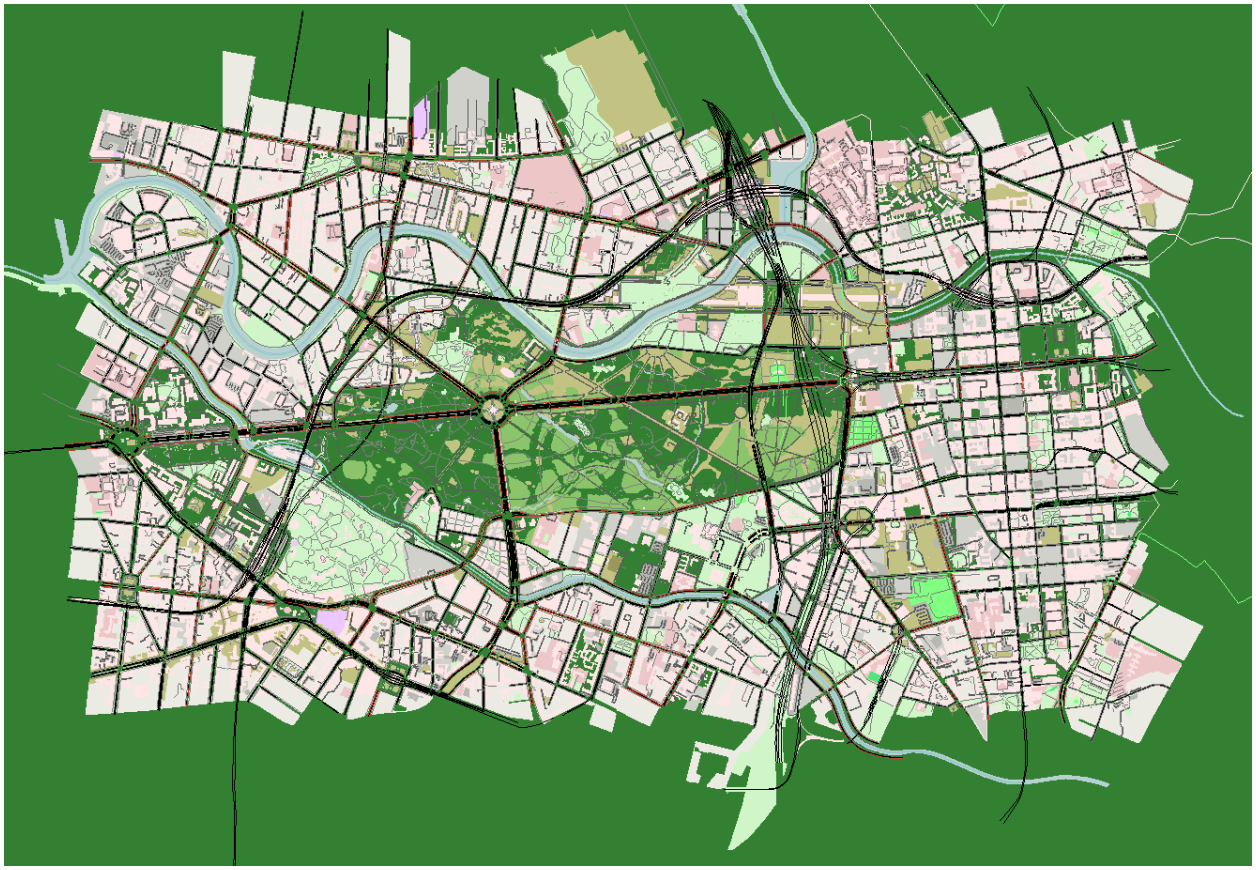


As described in the instructions we create a scenario with a number of cars, bikes, pedestrians and ships.

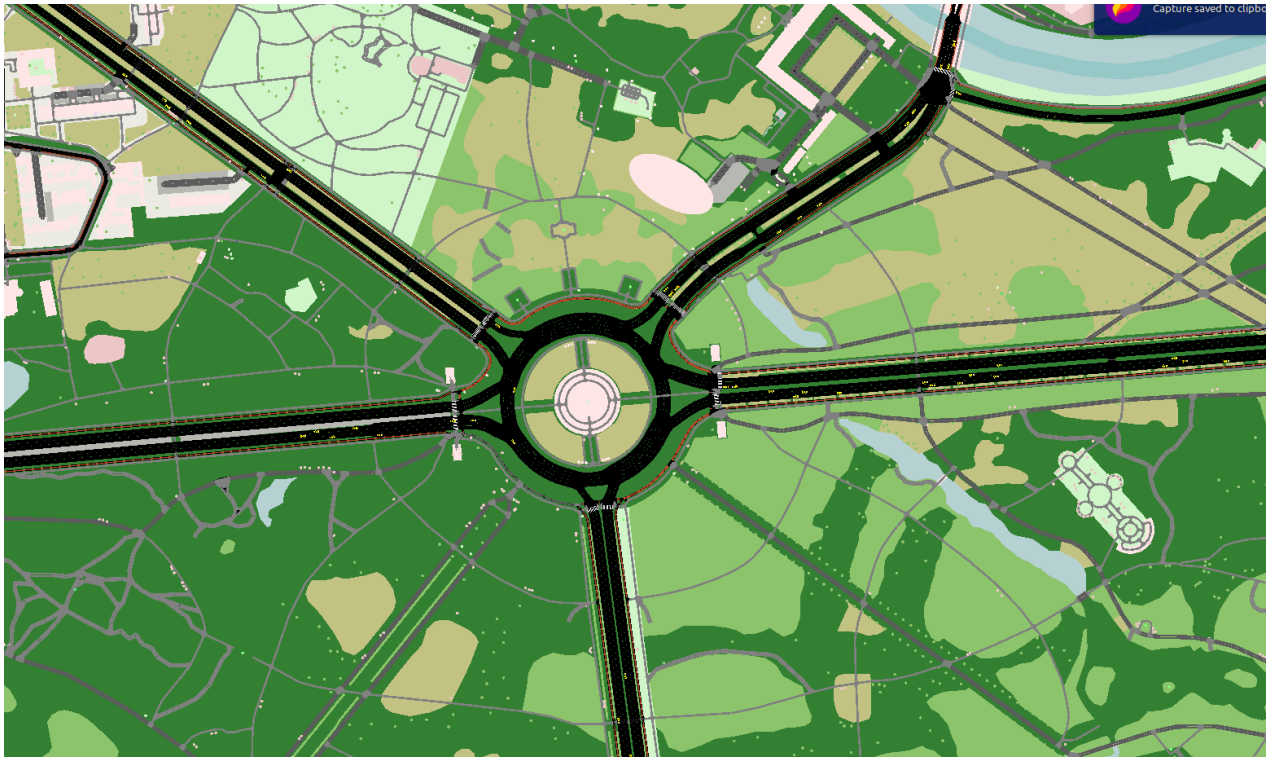


Once processed the map and the vehicles / people appear in the SUMO GUI application:

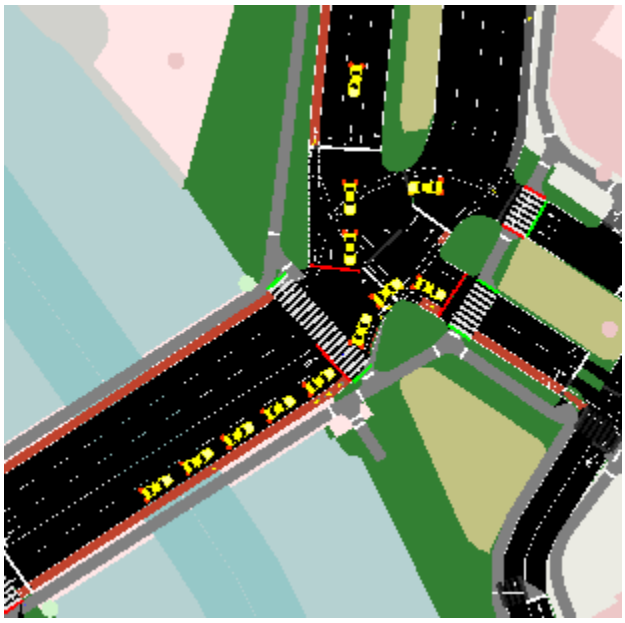
```
Loading net-file from '/home/zwerg/Sumo/2023-04-02-21-46-14/osm.net.xml.gz' ...
```

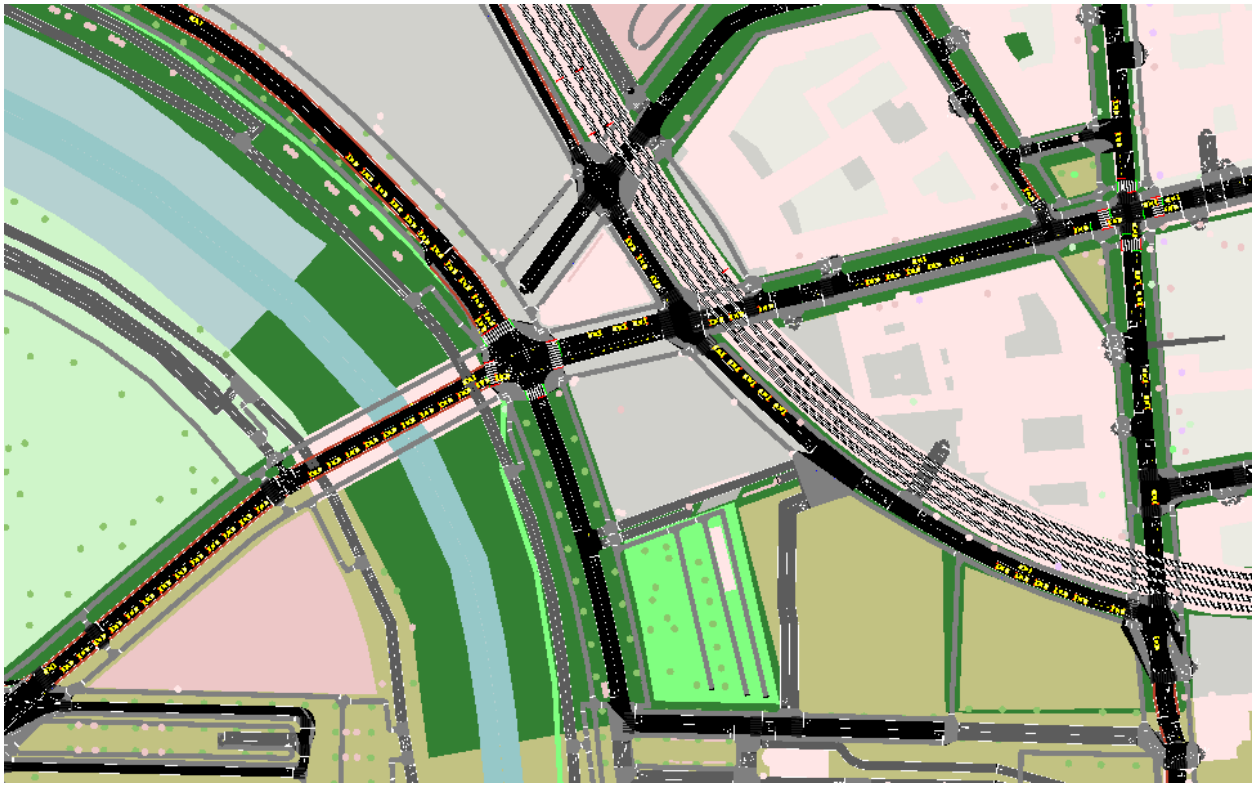


There are little yellow cars happily zipping by:

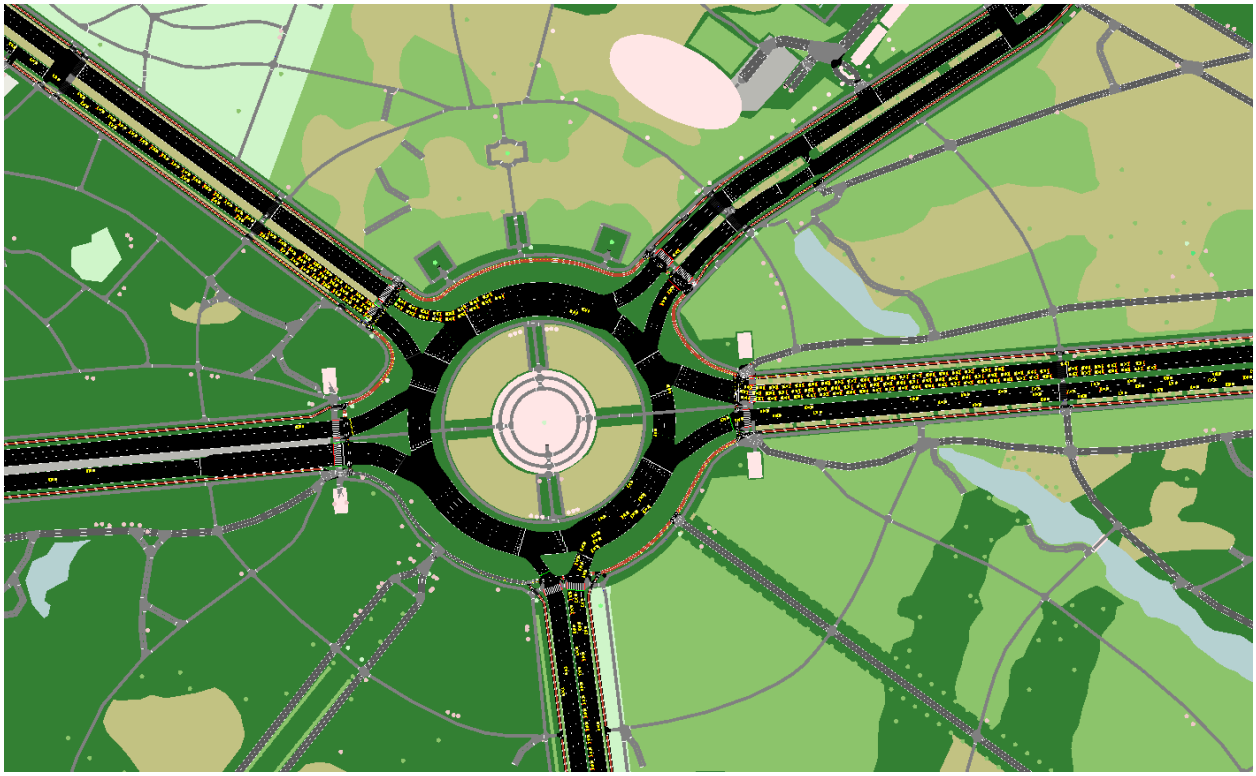


Some areas get crowded as cars have to wait for traffic lights:

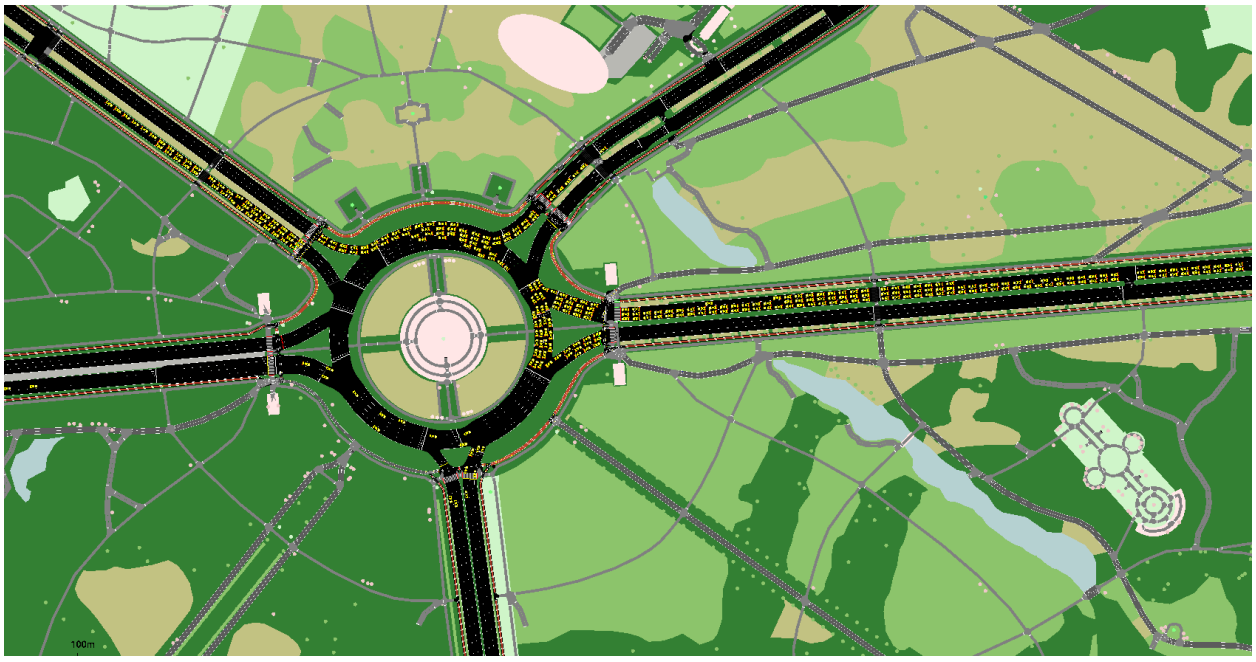




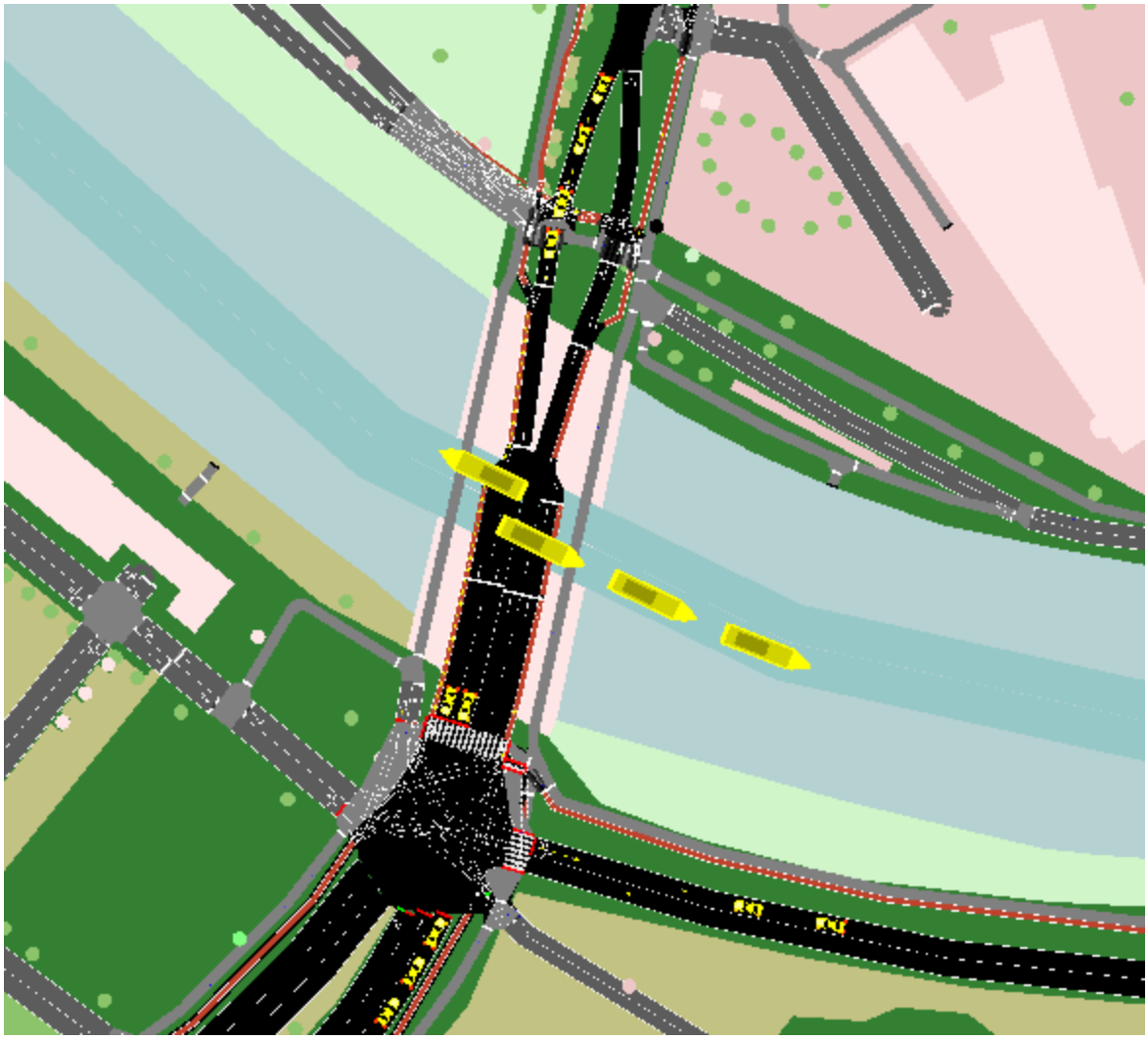
After about 2400 seconds into the simulation also the main hubs (roundways) get crowded:



After about 2900 seconds there is a jam:



Boat traffic is a bit more lenient



The simulation ran for 3600 seconds.

We obtain a folder with the results and decompress `osm.net.xml.gz` for parsing it using `parser.py`:

```
python parser.py --fcd fcd.xml --queue queue.xml --network
osm.net.xml
```

.

```
total 81940
drwxr-xr-x 2 zwerg input     4096 Apr  2 22:13 .
drwxr-xr-x 4 zwerg input     4096 Apr  2 21:46 ..
-rwxr-xr-x 1 zwerg input     1286 Apr  2 21:49 build.bat
-rw-r--r-- 1 zwerg input  5297970 Apr  2 21:46 osm_bbox.osm.xml.gz
-rw-r--r-- 1 zwerg input   754411 Apr  2 21:47 osm.bicycle.trips.xml
-rw-r--r-- 1 zwerg input     1536 Apr  2 21:46 osm.netccfg
-rw-r--r-- 1 zwerg input 60325430 Apr  2 21:46 osm.net.xml
-rw-r--r-- 1 zwerg input  8562469 Apr  2 21:46 osm.net.xml.gz
-rw-r--r-- 1 zwerg input   964940 Apr  2 21:48 osm.passenger.trips.xml
-rw-r--r-- 1 zwerg input  4682581 Apr  2 21:49 osm.pedestrian.rou.xml
-rw-r--r-- 1 zwerg input   977955 Apr  2 21:48 osm.pedestrian.trips.xml
-rw-r--r-- 1 zwerg input      672 Apr  2 21:46 osm.polycfg
-rw-r--r-- 1 zwerg input  2279206 Apr  2 21:46 osm.poly.xml.gz
-rw-r--r-- 1 zwerg input    11226 Apr  2 21:49 osm.ship.trips.xml
-rw-r--r-- 1 zwerg input      975 Apr  2 21:49 osm.sumocfg
-rw-r--r-- 1 zwerg input       88 Apr  2 21:49 osm.view.xml
-rwxr-xr-x 1 zwerg input       23 Apr  2 21:49 run.bat
```

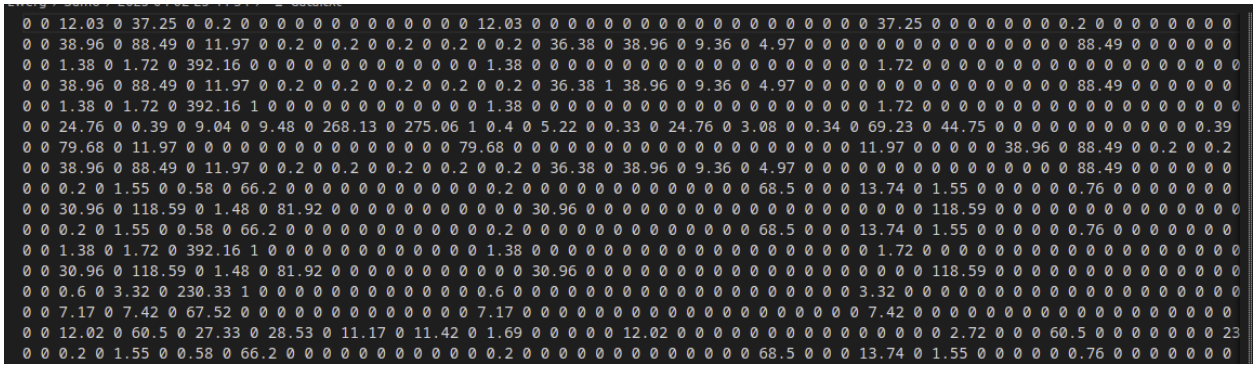
Appendix 2: Predict the traffic hotspot

Now we need to extract the information from the simulated data using:
`sumo -c osm.sumocfg --collision.action warn --time-to-teleport -1 --fcd-output fcd.xml --queue-output queue.xml`

During the extraction we get a lot of warnings about jams, collisions and emergencies hinting at the drama which has played out on that simulated day in Berlin:

```
Warning: Person 'ped623' is jammed on edge ':10298357736_w0', time=1130.00.
Warning: Person 'ped1280' is jammed on edge ':26734302_w0', time=1130.00.
Warning: Vehicle 'veh2271' performs emergency braking on lane ':3366747771_0_3' with decel=9.00, wished=4.50,
 severity=1.00, time=1135.00.
Warning: Person 'ped426' is jammed on edge ':1338117984_w0', time=1141.00.
Warning: Vehicle 'bike2050' collision with person 'ped320', lane='-376995434_0', gap=-0.08, time=1141.00, sta
ge=move.
Warning: Person 'ped400' is jammed on edge ':26734302_w0', time=1142.00.
Warning: Person 'ped413' is jammed on edge ':10298357736_w0', time=1147.00.
Warning: Vehicle 'veh1374' performs emergency braking on lane ':267980710_2_1' with decel=9.00, wished=4.50,
severity=1.00, time=1153.00.
Warning: Person 'ped1271' is jammed on edge ':cluster_1326766121_1326766239_968049115_w0', time=1154.00.
Warning: Person 'ped1015' is jammed on edge ':cluster_100532362_26734234_3336091666_4603782021_#4more_w0', ti
me=1158.00.
```

After parsing using parser.py the feature file data.txt and the labels.txt get created. The features (cars waiting at crossings) look like so:



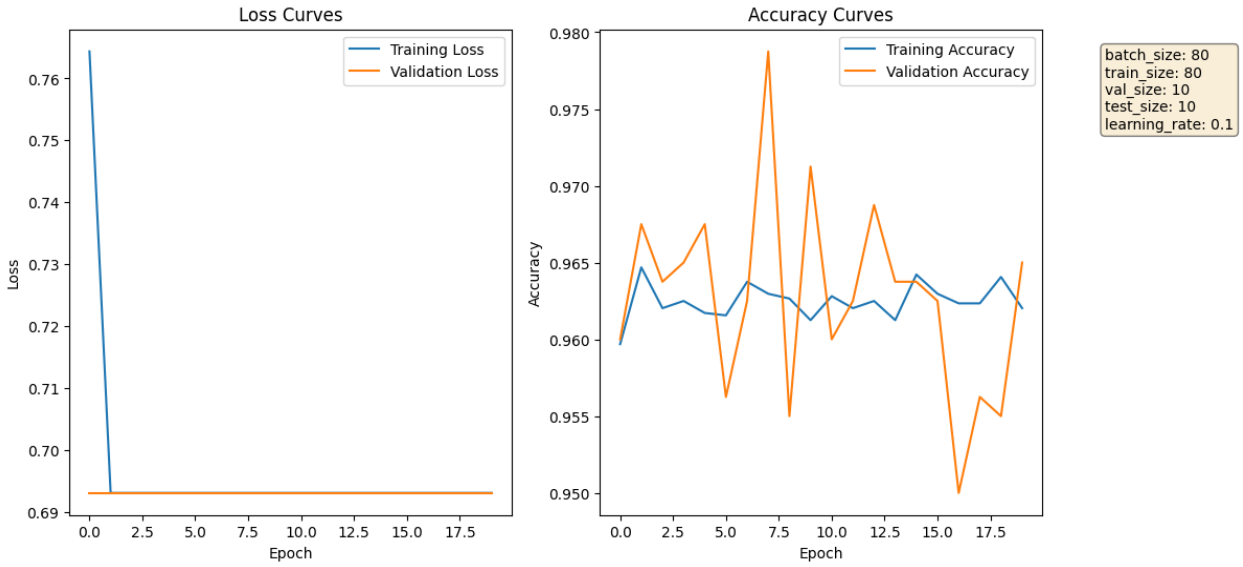
Labels.txt gives a binary label [0,1] for each row.

We now build a classifier using Tensorflow.
The classifier is a multilayer perceptron with 3-4 layers.

Small size area

The simulation of the small size area, which yielded a dataset size similar to the sample data, produced a model with about 97% test accuracy. The model accuracy appeared to plateau after 1-2 epochs. Hence, making the model deeper (4 rather than 3 layers) did not help with increasing the accuracy.

Two layers:



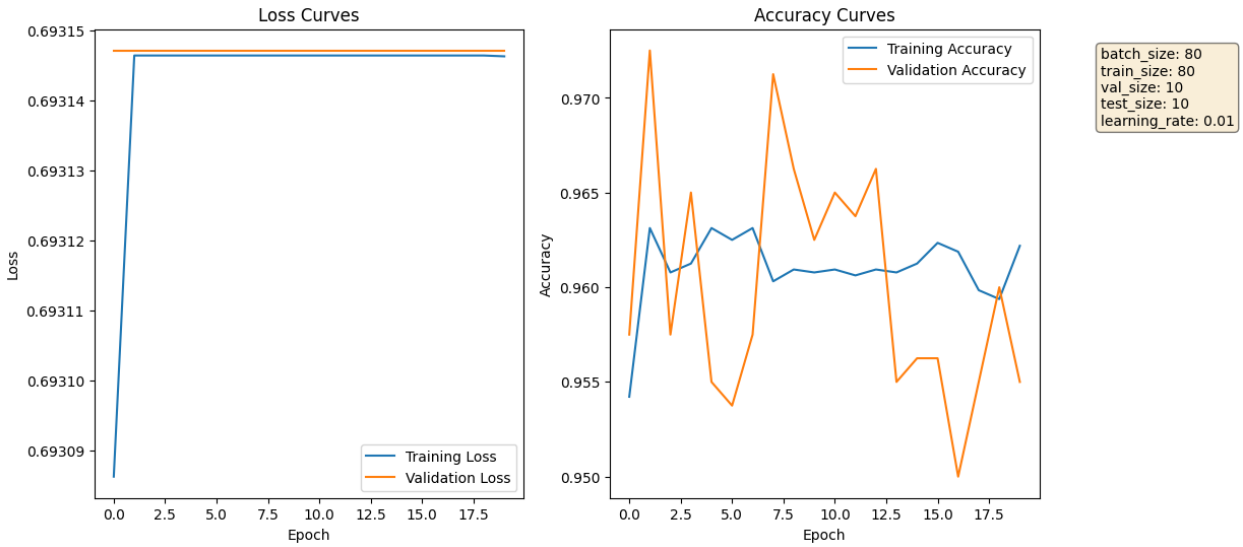
Layer (type)	Output Shape	Param #
Hidden1 (Dense)	(None, 80)	16080

Hidden2 (Dense)	(None, 20)	1620
Output (Dense)	(None, 2)	42

==========

Total params: 17,742
Trainable params: 17,742
Non-trainable params: 0

Three layers:




```
## Test set
# test loss, test accuracy, test mse
model.evaluate(test_dataset)
```

```
10/10 [==============================] - 1s 13ms/step - loss: 0.6931 - accuracy: 0.9725 - mse: 0.0262
[0.6931471824645996, 0.9725000262260437, 0.026249999180436134]
```

Large size area

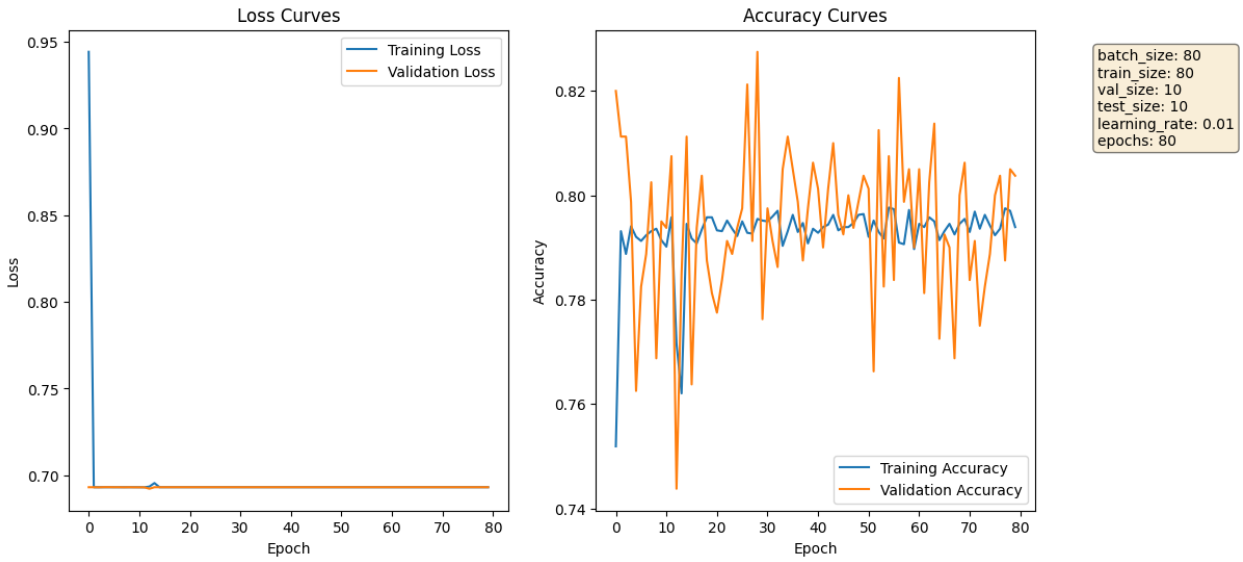
The simulation on the larger area only produced models with up to about 80 % test accuracy, in spite of our efforts of making it wider and deeper or to use a convolutional model with dropout.

Variant 1: same model as before

Layer (type)	Output Shape	Param #

Hidden1 (Dense)	(None, 80)	16080
Hidden2 (Dense)	(None, 40)	3240
Hidden3 (Dense)	(None, 20)	820
Output (Dense)	(None, 2)	42

Total params: 20,182
Trainable params: 20,182
Non-trainable params: 0



Test loss, test accuracy, test mse
[0.6931471824645996, 0.7875000238418579, 0.21250000596046448]

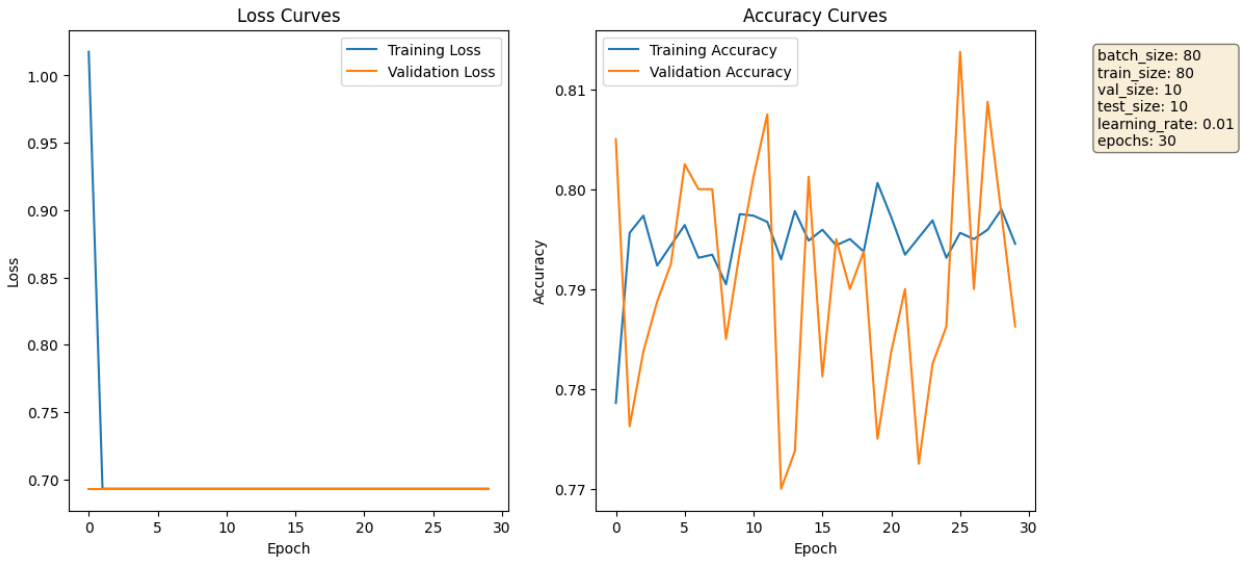
Variant 2: wider and deeper model

Layer (type)	Output Shape	Param #
Hidden1 (Dense)	(None, 240)	48240
Hidden2 (Dense)	(None, 120)	28920
Hidden3 (Dense)	(None, 60)	7260

Hidden4 (Dense)	(None, 15)	915
Output (Dense)	(None, 2)	32

==========

Total params: 85,367
Trainable params: 85,367
Non-trainable params: 0



[0.6931471824645996, 0.8012499809265137, 0.19875000417232513]
The wider model produce slightly higher test accuracy than the previous one.

Variant 3: CNN model

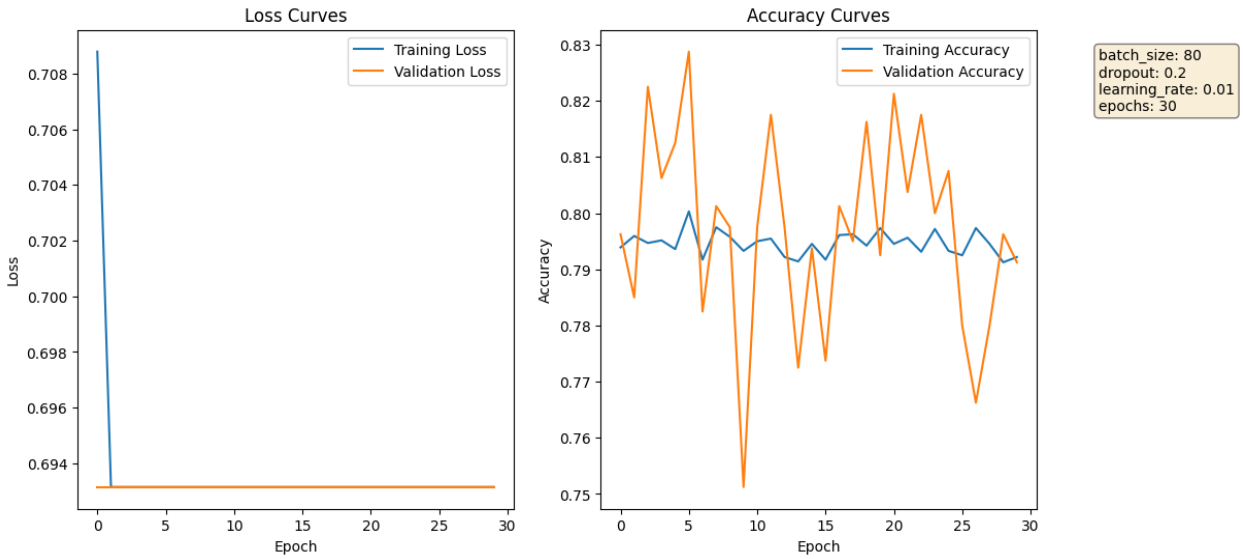
Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 200, 1)]	0
Conv1 (Conv1D)	(None, 97, 200)	1600
Conv2 (Conv1D)	(None, 95, 100)	60100
flatten (Flatten)	(None, 9500)	0
Hidden2 (Dense)	(None, 100)	950100

dropout_3 (Dropout)	(None, 100)	0
Hidden3 (Dense)	(None, 60)	6060
Hidden4 (Dense)	(None, 15)	915
Output (Dense)	(None, 2)	32

==========

Total params: 1,018,807
Trainable params: 1,018,807
Non-trainable params: 0

_____



[0.6931471824645996, 0.7850000262260437, 0.2150000035762787]

The convolutional model did not do better than the smaller MLP model. This might be because there is some noise in the data or perhaps the conversion of simulation results with the parser needs to be performed differently. We couldn't go into all detail on that.
Suffice it to say, if the number of crossings and cars remains small we can predict traffic jams with high accuracy. Predicting jams for an entire city using the above method would, however, not be precise. It may give insights into hot spots of traffic jams which could be used by civil engineers to re-route traffic to lesser used routes.