nuplan-devkit

Release v0.1

patk

May 17, 2023

CONTENTS

Note: The nuPlan Planning Competition is live! Submission Deadline extended to May 26th, 2023.

CHAPTER

ONE

NUPLAN

The world's first benchmark for autonomous vehicle planning.

1.1 Sensor Data Release

1.1.1 IMPORTANT: The file structure has changed! Please check Dataset Setup page for the updated file structure.

- The nuPlan sensor data for the v1.1 dataset has been released. Please download the latest dataset from the nuPlan page.
- Due to the size of the sensor data, it will be released gradually. This first set of sensor data are the blobs corresponding to nuPlan mini.
- A short tutorial for the sensor data is provided `nuplan_sensor_data_tutorial.ipynb` to get you started.

1.2 Planning challenges

1.2.1 IMPORTANT: The base docker image used in nuPlan submission has been updated. You should rebuild your submission container with the new `Dockerfile.submission`

- The Planning Challenge will be using devkit version 1.2 from now on. Submissions generated from version v1.1 should remain compatible. However, please double-check by submitting to the warm-up phase.
- The challenge will be presented as part of the End-to-End Autonomous Driving workshop at CVPR 2023
- The nuPlan Dataset v1.1 has been released. Please download the latest dataset from the nuPlan page.

1.3 Changelog

- May 11th 2023
  - v1.2.2 Devkit: Upated the submission base images.
- May 9th 2023
  - v1.2.1 Devkit: Update to competition dates. Submission Deadline extended to May 26th, 2023.
- April 25th 2023
  - v1.2 Devkit: The nuPlan sensor data have been released! Improved feature caching and nuBoard dashboard functionality. Changed dataset file structure, data interfaces now allow retrieval of sensor data. Pinned several packages including hydra, numpy and sqlalchemy.
- January 20th 2023
  - v1.1 Devkit: The official nuPlan Challenge Release. Optimized training caching, simulation improvements, shapely 2.0 update.
- Oct 13th 2022
  - v1.1 Dataset: Full nuPlan dataset - improved route plan, traffic light status, mission goal and more!
  - v1.0 Devkit: Update to nuplan-v1.1 dataset, metrics improvements, route plan fixes, documentation, IDM-Planner
- Sep 09 2022
  - v0.6 Devkit: Smart agents optimizations, nuBoard improvements, metrics improvements, submission pipeline deployment and documentation.
- Aug 26 2022
  - v0.5 Devkit: New map features, simulation improvements, open-loop detections with smart agents, iLQR tracker, metrics improvements and documentation.
- Aug 05 2022
  - v0.4 Devkit: Database optimization, PYGEOS warning suppression, metrics improvements, scenario filter for training.
- Jul 15 2022
  - v0.3 Devkit: Updates to the devit including but not limited to: nuBoard update, reduce RAM usage during caching and training, new map APIs, simulation and metrics runtime improvements.
- Jun 10 2022
  - v1.0 Dataset: Full nuPlan dataset release with over 1,300 hours of driving data (15,000+ logs) across 4 cities (Las Vegas, Pittsburgh, Boston, Singapore).
  - v0.2 Devkit: Multiple devkit updates with improvements across the whole framework (planning models, training, simulation, metrics, dashboard, tutorials and more).
- Dec 19 2021
  - v0.2 Dataset: Fixed bugs in the teaser nuPlan dataset.
- Dec 10 2021
  - v0.1 Dataset: Initial teaser nuPlan dataset release with over 200 hours of driving data (350+ logs) across Las Vegas.
  - v0.1 Devkit: Initial nuPlan devkit release.

1.4 Devkit and dataset setup

Please refer to the installation page for detailed instructions on how to setup the devkit.

Please refer to the dataset page for detailed instructions on how to download and setup the dataset.

1.5 Getting started

For those interested in participating in the nuPlan Planning Competition, please refer to the competition landing page.

Please follow these steps to make yourself familiar with the nuPlan dataset:

- Familiarize yourself with the main features of nuPlan and the dataset description.
- Setup the devkit and dataset as described *above*.
- Walk through the tutorials in this folder or run them yourself using `jupyter notebook ~/nuplan-devkit/tutorials/<filename>.ipynb` and replacing `<filename>` with the tutorial's name. The following tutorials are available:
  - `nuplan_framework.ipynb`: Main tutorial for anyone who wants to dive right into ML planning. It describes how to 1) train an ML planner, 2) simulate it, 3) measure the performance and 4) visualize the results.
  - `nuplan_scenario_visualization.ipynb`: Tutorial for visualizing various scenario types contained within the nuPlan dataset (e.g. unprotected turns, lane changes, interactions with pedestrians and more).
  - `nuplan_planner_tutorial.ipynb`: Tutorial with instructions on how to develop and simulate a planner from scratch within the nuPlan framework.
  - `nuplan_advanced_model_training.ipynb`: This notebook will cover the details involved in training a planning model in the NuPlan framework. This notebook is a more detailed deep dive into the NuPlan architecture, and covers the extensibility points that can be used to build customized models in the NuPlan framework.
- Familiarize yourself with the nuPlan CLI, which gets installed by installing the devkit using `pip` (editable and not) by running:

```
nuplan_cli --help
nuplan_cli COMMAND --help
```

- Read the nuPlan paper to understand the details behind the dataset.

1.6 Performance tuning guide

Training configurations are important to ensure your expected system performance, for example preprocessing cost, training speed, and numerical stability. If you encounter problems related to aforementioned aspects, please refer to performance tuning guide to find potential solutions.

1.7 Devkit structure

Our code is organized in these directories:

```
nuplan_devkit
├── ci                - Continuous integration code - not relevant for average users.
├── docs              - Readmes and other documentation of the repo and dataset.
├── nuplan            - The main source folder.
│   ├── common        - Code shared by "database" and "planning".
│   ├── database      - The core devkit used to load and render nuPlan dataset and maps.
│   ├── planning      - The stand-alone planning framework for simulation, training and␣
↪evaluation.
│   ├── submission    - The submission engine used for the planning challenge.
│   └── cli           - Command line interface tools for the nuPlan database.
└── tutorials         - Interactive tutorials, see "Getting started".
```

1.8 Citation

Please use the following citation when referencing nuPlan:

```
@INPROCEEDINGS{nuplan,
  title={NuPlan: A closed-loop ML-based planning benchmark for autonomous vehicles},
  author={H. Caesar, J. Kabzan, K. Tan et al.,},
  booktitle={CVPR ADP3 workshop},
  year=2021
}
```

CHAPTER

TWO

INSTALLATION

We provide step-by-step instructions to install the nuPlan devkit. For a high-level overview, please read the general readme first.

- *Download*
- *Install Python*
- *Install virtual environment*
- *Install the devkit*
- *Install required packages*
- *Setup environment variables*

2.1 Download the devkit

Download the devkit and move inside the folder:

```
cd && git clone https://github.com/motional/nuplan-devkit.git && cd nuplan-devkit
```

The above will download the files to your home directory. While you can change this to an arbitrary directory, the rest of our tutorials assumes that you are using the home directory.

2.2 Install Python

The devkit is tested for Python 3.9 on Ubuntu.

- For **Ubuntu**: If the right Python version is not already installed on your system, install it by running:

```
sudo apt install python-pip
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.9
sudo apt-get install python3.9-dev
```

- For **Mac OS** download and install from `https://www.python.org/downloads/mac-osx/`.

2.3 Install virtual environment

Next we setup a virtual environment. We recommend Conda for this purpose.

2.3.1 Install miniconda

See the official Miniconda page.

2.3.2 Create a Conda environment

We create a new Conda environment using environment.yml.

```
conda env create -f environment.yml
```

2.3.3 Activate the environment

If you are inside the Conda environment, your shell prompt should look like: `(nuplan) user@computer:~$` Going forward, we shall always assume the Conda environment is active. If that is not the case, you can enable the virtual environment using:

```
conda activate nuplan
```

To deactivate the virtual environment, use:

```
conda deactivate
```

2.4 Install the devkit

2.4.1 Option A: Install PIP package from remote

Note: This option is not yet supported. It will be added soon.

For beginners, the easiest option is to install the PIP package:

```
pip install nuplan-devkit
```

This installs the devkit and all of its dependencies.

2.4.2 Option B: Install PIP package from local

We recommend that you install the local devkit as a PIP package:

```
pip install -e .
```

This installs the devkit and all of its dependencies. Note that the editable mode (`-e`) is optional and means that the code is used in-place, rather than being copied elsewhere, and can be modified for easy development.

2.4.3 Option C: Run source code directly

Alternatively, if you don't want to use the pip package, you can manually add the `nuplan-devkit` directory to your `PYTHONPATH` environmental variable, by adding the following to your `~/.bashrc`:

```
export PYTHONPATH="${PYTHONPATH}:$HOME/nuplan-devkit"
```

To activate these changes you then need to run:

```
source ~/.bashrc
```

To install the dependencies, run the following command:

```
pip install -r requirements_torch.txt
pip install -r requirements.txt
```

2.5 Change default directories

As described in the general readme, the default nuPlan directories are:

```
~/nuplan/dataset    -   The dataset folder. Can be read-only.
~/nuplan/exp        -   The experiment and cache folder. Must have read and write access.
```

If you want to change these on your system, you need to set the corresponding environment variables in your `~/.bashrc`, e.g.:

```
export NUPLAN_DATA_ROOT="$HOME/nuplan/dataset"
export NUPLAN_MAPS_ROOT="$HOME/nuplan/dataset/maps"
export NUPLAN_EXP_ROOT="$HOME/nuplan/exp"
```

This step is also required if you want to run any of the unit tests in the devkit.

That's it you should be good to go!

CHAPTER

THREE

DATASET SETUP

3.1 Download

To download nuPlan you need to go to the Download page, create an account and agree to the Terms of Use. After logging in you will see multiple archives. For the devkit to work you will need to download *all* archives. Please unpack the archives to the `~/nuplan/dataset` folder.

3.2 Filesystem hierarchy

The final hierarchy should look as follows (depending on the splits downloaded above):

```
~/nuplan
├── exp
│   └── ${USER}
│       ├── cache
│       │   └── <cached_tokens>
│       └── exp
│           └── my_nuplan_experiment
└── dataset
    ├── maps
    │   ├── nuplan-maps-v1.0.json
    │   ├── sg-one-north
    │   │   └── 9.17.1964
    │   │       └── map.gpkg
    │   ├── us-ma-boston
    │   │   └── 9.12.1817
    │   │       └── map.gpkg
    │   ├── us-nv-las-vegas-strip
    │   │   └── 9.15.1915
    │   │       └── map.gpkg
    │   └── us-pa-pittsburgh-hazelwood
    │       └── 9.17.1937
    │           └── map.gpkg
    └── nuplan-v1.1
        ├── splits
        │   ├── mini
        │   │   ├── 2021.05.12.22.00.38_veh-35_01008_01518.db
        │   │   ├── 2021.06.09.17.23.18_veh-38_00773_01140.db
        │   │   ├── ...
```

(continues on next page)

(continued from previous page)

```
        │   │   └── 2021.10.11.08.31.07_veh-50_01750_01948.db
        │   └── trainval
        │       ├── 2021.05.12.22.00.38_veh-35_01008_01518.db
        │       ├── 2021.06.09.17.23.18_veh-38_00773_01140.db
        │       ├── ...
        │       └── 2021.10.11.08.31.07_veh-50_01750_01948.db
        └── sensor_blobs
            ├── 2021.05.12.22.00.38_veh-35_01008_01518
            │   ├── CAM_F0
            │   │   ├── c082c104b7ac5a71.jpg
            │   │   ├── af380db4b4ca5d63.jpg
            │   │   ├── ...
            │   │   └── 2270fccfb44858b3.jpg
            │   ├── CAM_B0
            │   ├── CAM_L0
            │   ├── CAM_L1
            │   ├── CAM_L2
            │   ├── CAM_R0
            │   ├── CAM_R1
            │   ├── CAM_R2
            │   └──MergedPointCloud
            │       ├── 03fafcf2c0865668.pcd
            │       ├── 5aee37ce29665f1b.pcd
            │       ├── ...
            │       └── 5fe65ef6a97f5caf.pcd
            │
            ├── 2021.06.09.17.23.18_veh-38_00773_01140
            ├── ...
            └── 2021.10.11.08.31.07_veh-50_01750_01948
```

If you want to use another folder, you can set the corresponding environment variable.

CHAPTER

FOUR

NUPLAN PLANNING CHALLENGE

4.1 Overview

The main focus of the nuPlan planning challenge is to evaluate a motion planning system in realistic driving scenarios, using multiple performance metrics. In this challenge, it is assumed that a planner will consume a top-down semantic representation of detected traffic participants (vehicles, bicycles, etc.) and static obstacles and plan the vehicle's future trajectory for a specific time horizon. The challenge is organized into three increasingly complex modes:

1. Open-loop
2. Closed-loop non-reactive agents
3. Closed-loop reactive agents

4.2 Competition Timeline

- January 30th, 2023 - Warm-up phase and Test phase opens for submission
- May 26th, 2023 - Competition closes - Finalists are announced and invited to share their code with us for verification
- June 2th, 2023 - Winners are announced
- June 18th, 2023 - The competition results are presented at the End-to-End Autonomous Driving workshop at CVPR 2023

4.3 How to enter

4.3.1 Getting Started

1. Sign up and download the dataset from https://www.nuscenes.org/nuplan.
2. Clone the devkit. Feel free to ask any questions by raising an issue on our GitHub: https://github.com/motional/nuplan-devkit
3. Create a team on EvalAI.

4.3.2 Making a submission

1. Go through the nuPlan submission tutorial. The tutorial will show how to create a valid Docker image.
2. Submit your Docker image through EvalAI.
3. Send the following information to nuScenes@motional.com after submitting on EvalAI

```
- Team name
- Method name
- Authors
- Affiliations
- Country (Of the individual or team's institution)
- Method description (5+ sentences)
- Project URL (If applicable)
- Paper URL (If applicable)
- Submission ID
```

4.3.3 Evaluation protocol and rules

Participants of the nuPlan planning challenge will submit their code to the challenge's cloud-based evaluation server through EvalAI - an open-source evaluation platform. The code will be submitted in the form of a Docker image and will be executed for each of the challenge modes (open-loop, non-reactive closed-loop and reactive closed-loop simulation) against the hidden test set of the challenge.

After submission, metrics that were generated for each simulation will be aggregated and sent to the public leaderboard. A final scoring function will be used to rank all submissions in each challenge. In addition, granular metrics and error analysis plots will be available for each submission on the challenge's page of the nuPlan official website (coming soon). For each of the three planning challenges, the following set of rules applies:

- A fixed set of scenario simulations for each scenario type (e.g. lane change) will be run to evaluate a submission.
- Each scenario simulation will start from a fixed point in time that represents the beginning of the scenario (e.g. the start of an unprotected turn).
- The current and past (up to 2s) scene information for each simulation iteration will be passed to the planner - that includes ego pose, ego route, agents, and static/dynamic map information.
- Each planner will have a fixed time budget of 1s for each simulation iteration, after which the simulation will time out.
- The simulation horizon will be up to 15s for each scenario and the simulation frequency will be 10Hz.
- There will be a strict limit of three submissions for each participant for this competition.
- Every submission should provide a short description of the method used.
- Submissions that obfuscate code in the submitted Docker image will not be considered for the competition and therefore winning prizes. However, participants are still welcome to submit their "top secret" planner
- The top performing submissions will be inspected for compliance with the challenge's rules - any attempt to circumvent these rules will result in a permanent ban of the team or company from all future nuPlan challenges.

4.4 Phases

4.4.1 Warm-up Phase

The warm-up phase allows participants to test the submission pipeline. During this phase, participants may submit up to 10 submissions. The submissions will be run on all three challenges. The scenario types used for the warm-up phase are the same as the ones that will be used for the test phase. However, the number of scenarios run will be smaller, and the data is not taken from the test split. The metrics and the overall scores will be displayed on the leaderboard.

Start: January 30th, 2023

End: May 26th, 2023

4.4.2 Test Phase

The results from the test phase will be used to determine the winners of each challenge. All submissions will be evaluated on a larger set of scenarios from the test split. The scores from the test phases will be used to shortlist finalists. Finalists will then be invited to share their code with the organizers. Once the submissions from the finalists are verified by the organizers, the same scores are used to determine the winners of each challenge. Participants will be allowed to submit a maximum of three submissions.

Start: January 30th, 2023

End: May 26th, 2023

4.5 Challenges

4.6 Planner Output Requirements

Warning: The following requirements must be met for a submission to be considered valid. Failing to meet these requirements will result in your submission failing.

Planner Trajectory Requirements	Value
Expected minimum horizon length	8s
Expected minimum horizon steps	2
Expected signals	x position in global frame of Ego's rear axles center y position in global frame of Ego's rear axles center heading in global frame time stamp

4.7 Evaluation server

Configuration	Value
Simulation iteration time budget	1s
Max submissions per each user/team	3
Instance type	16 CPU, 64 RAMResource partitioning: SimulationContainer: 0GPU, 2vCPU, 28GB RAMSubmissionContainer: 1GPU, 8vCPU, 32GB RAM

All challenges will be run with the following common configurations

Common Challenge Configuration	Value
Frequency	10Hz
Rollout horizon	15s
Past observation horizon	2s

4.7.1 Challenge 1: Open-loop

In this challenge, the planning system is tasked with imitating the expert driver in open-loop. For every time step, the planner's predicted trajectory is scored based on a set of predefined open-loop metrics. Namely, it is compared directly against the driven ground-truth trajectory. As the name suggests, the planned trajectory will not be used to alter the state of the simulation.

Configuration	Value
Metric computation horizon	3s, 5s, 8s @ 1Hz sampling
Controller	N/A
Observations	VehiclesPedestriansCyclistsGeneric ObjectsTraffic conesBarriersConstruction Zone Signs

Scoring

The overall score of challenge 1 considers the following factors:

- Average Displacement Error (ADE)
- Final Displacement Error (FDE)
- Average Heading Error (AHE)
- Final Heading Error (FHE)
- Miss Rate

The details of each metric can be found here. The details of the scoring hierarchy can be found here

4.7.2 Challenge 2: Closed-loop non-reactive agents

In this challenge, the planner outputs a planned trajectory using the information available at each time step, similar to the previous case. The predicted trajectory is used as a reference for a tracking controller to simulate the vehicle's state at the next time step. Nonetheless, the other agents are replayed from a log according to their observed states in open-loop - hence, the name non-reactive. Most importantly, the ground-truth is no longer applicable to the evaluation of the planner. Instead, a suite of closed-loop metrics will be used to judge the performance of a planner

Configuration	Value
Controller	LQR
Observations	VehiclesPedestriansCyclistsGeneric ObjectsTraffic coneBarriersConstruction Zones Signs

Scoring

The overall score of challenge 2 considers the following factors:

- At-fault collision
- Drivable area compliance
- Driving direction compliance
- Making progress
- Time to collision (TTC)
- Speed limit compliance
- Ego progress along the expert's route ratio
- Comfort

The details of each metric can be found here. The details of the scoring hierarchy can be found here

4.7.3 Challenge 3: Closed-loop reactive agents

In the final and most complex challenge, the vehicles in the scene become reactive. In other words, the vehicles will react to the actions of the planner-controlled vehicle along with the behavior of other agents, the policy determining the behavior of agents is an Intelligent Driver Model (IDM) policy. The rest of the observations are still replayed in open-loop. Similar to the non-reactive closed-loop challenge, a suite of metrics will be used to judge the performance of a planner in closed-loop.

Configuration	Value
Controller	LQR
Observations	Reactive:VehiclesOpen-loopPedestriansGeneric ObjectsTraffic coneBarriersConstruction Zones Signs

Scoring

The overall score of challenge 3 considers the following factors:

- At-fault collision
- Drivable area compliance
- Driving direction compliance
- Making progress
- Time to collision (TTC)
- Speed limit compliance
- Ego progress along the expert's route ratio
- Comfort

The details of each metric can be found here. The details of the scoring hierarchy can be found here

4.8 Scenario Selection

All three challenges will be run on the following scenario types.

Scenario Type	Description
starting_straight_traffic_light_intersection_traversal	Ego at the start of a traversal going straight across an intersection area controlled by traffic lights while not stopped
high_lateral_acceleration	Ego high ego acceleration (1.5 < acceleration < 3 m/s^2) across the lateral axis with high yaw rate while not turning
changing_lane	Ego at the start of a lane change towards an adjacent lane
high_magnitude_speed	Ego high velocity magnitude with low acceleration (velocity > 9 m/s)
low_magnitude_speed	Ego low ego velocity magnitude (0.3 < velocity < 1.2 m/s) with low acceleration while not stopped
starting_left_turn	Ego at the start of a traversal turning left across an intersection area while not stopped
starting_right_turn	Ego at the start of a traversal turning right across an intersection area while not stopped
stopping_with_lead	Ego starting to decelerate (acceleration magnitude < -0.6 m/s^2, velocity magnitude < 0.3 m/s) with a leading vehicle ahead (distance < 6 m) at any area
following_lane_with_lead	Ego following (velocity > 3.5 m/s) its current lane with a moving leading vehicle ahead on the same lane (velocity > 3.5 m/s, longitudinal distance < 7.5 m)
near_multiple_vehicles	Ego nearby (distance < 8 m) of multiple (>6) moving vehicles while ego is moving (velocity > 6 m/s)
traversing_pickup_dropoff	Ego during the traversal of a pickup/drop-off area while not stopped
behind_long_vehicle	Ego behind (3 m < longitudinal distance < 10 m) a long (length > 8 m) vehicle in the same lane as ego (lateral distance < 0.5 m)
waiting_for_pedestrian_to_cross	Ego waiting for a nearby (distance < 8 m, time to intersection < 1.5 m) pedestrian to cross a crosswalk area while ego is not stopped and the pedestrian is not at a pickup/drop-off area
stationary_in_traffic	Ego is stationary with multiple (>6) vehicles nearby (distance < 8 m)

4.9 How to win

The submissions are ranked based on the mean overall score across all three challenges.

4.9.1 Prizes

- 1st place: USD $10,000
- 2nd place: USD $8,000
- 3rd place: USD $5,000
- Innovation Prize: USD $5,000
  - The Innovation Prize will be awarded to the most innovative submission judged by a panel of Motional experts.
- Internship opportunity
  - Eligible individual participants may be considered for internships at Motional based on their performance on the challenges. Any offers for internships are subject to the Competition Terms, the participant successfully completing the Motional application process, and any other terms and conditions solely determined by Motional.

Terms and conditions apply.

4.10 Contact

To contact the organizers, please make an issue on our GitHub page or email us at nuScenes@motional.com

CHAPTER

FIVE

SUBMISSION TUTORIAL

This tutorial will walk you through how to generate a valid submission, and send it to the evaluation server.

5.1 Prerequisites

In this guide, it is assumed you followed the `nuplan_framework` tutorial and are familiar with the nuPlan environment. It is suggested, but not required, to first go through this tutorial with the example Planner (`SimplePlanner`) and then make the necessary modifications to use your custom planner.

Another requirement is to have a local Docker installation. You can follow the official documentation here. Furthermore, for local testing `docker-compose` is needed, and a version >=1.28.0 is required. On a linux machine you can install the latest version (at the moment of writing) with:

```
sudo apt remove docker-compose
sudo curl -L "https://github.com/docker/compose/releases/download/v2.9.0/docker-compose-
↪$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
sudo chmod +x /usr/local/bin/docker-compose
```

5.1.1 Protocol specifications

The client/server communication is made using gRPC, the configuration files and code related to the communication can be found at `~/nuplan_devkit/nuplan/submission`. In order for submissions to work as expected, the protocol files (`protos/`) and the autogenerated (`challenge_pb2.py` and `challenge_pb2_grpc.py`) files **MUST NOT** be modified, as we will use our version of them.

Similarly, you should not modify `submission_container.py` and `submission_planner.py`, to avoid invalid submissions. You should modify the specified section of `entrypoint_submission.sh`.

5.2 Creating a valid submission

To be able to run your planner, the basic requirement is that it needs to inherit from AbstractPlanner, and implement the relevant interfaces.

Checkpoint Run a simulation with your custom planner using the run_simulation script, as exemplified in the tutorials.

Once you can run your planner, to be able to submit your planner for remote execution you will have to package it in a Docker container. Here are the steps:

1. Dockerfile.submission is the starting point. You might not need to edit it directly, however, feel free to change it as you see fit. If you need to edit the system dependencies you can do so by adding apt targets as needed in Dockerfile.submission. By default, anything under the `nuplan_devkit/nuplan` directory will be present in the Docker image at the absolute path `/nuplan_devkit/nuplan`.
2. To add additional pip requirements you may need, please append them to requirements_submission.txt.
3. You can copy checkpoints and other files inside your container, as long as they are in the same directory or in a subdirectory from `Dockerfile.submission`. To do so follow the example present in `Dockerfile.submission`.
4. run_submission_planner.py is where your planner is instantiated. This is done through hydra, which allows you to pass any arguments required to instantiate your planner from a config file.
   - For reference, see the SimplePlanner hydra config. For example, if you want to instantiate `SimplePlanner` but modify `max_velocity`, you could make a config file `simple_planner2.yaml` in `nuplan/planning/script/config/simulation/planner` containing:

     ```
     simple_planner2:
         _target_: nuplan.planning.simulation.planner.simple_planner.SimplePlanner
         _convert_: 'all'
         horizon_seconds: 10.0
         sampling_time: 0.25
         acceleration: [0.0, 0.0] # x (longitudinal), y (lateral)
         max_velocity: 7.0  # modified
         steering_angle: 0.0
     ```

     and then set `planner=simple_planner2` in `entrypoint_submission.sh`. More examples are resent in the other tutorials.
   - If you run into problems using hydra, you can instantiate the planner manually in run_submission_planner.py. See the included comments for pointers.

At this point, you need to make sure that you can generate a valid Docker image from the Dockerfile you edited.

Checkpoint Create a Docker image using the edited Dockerfile.submission. If you are unfamiliar with Docker, the command below should be a good starting point (to be run within `nuplan_devkit/`):

```
docker build --network host -f Dockerfile.submission . -t nuplan-evalservice-server:test.
↪contestant
```

Once you created the docker image, you need to test its behavior in the sever-client architecture, to make sure everything works fine. To do so you can use docker-compose which should take care of everything for you.

`docker-compose.yml` mounts the dataset and maps to the submission and simulation containers, as well as writing results from the simulation locally on your machine. By default, they mount inside the containers the directories where `$NUPLAN_DATA_ROOT`, `$NUPLAN_MAPS_ROOT` and `$NUPLAN_EXP_ROOT` point to, so you need to make sure these are set. The following command will create both the submission and the simulation images, and will start the simulation (check the entrypoint files for details).

Checkpoint Run the simulation by running the command below:

```
docker-compose up --build
```

You can inspect the results in your `$NUPLAN_EXP_ROOT` directory after simulation.

5.3 Submitting a solution

To submit through to EvalAI you will have to register on the EvalAI website. Then you will have to install the CLI and push your submission, with for example:

```
evalai push <image>:<tag> --phase random-dev-1856
```

If you run into problems, the exact command can be found on your competition submission page. The results will be visible on the EvalAI leaderboard.

CHAPTER

SIX

BASELINES

We provide several baselines within the devkit. These baselines are standard comparison points in which to compare new planners. Moreover, the baselines serve as a starting point for users to prototype their planner or simply tinker with it.

6.1 SimplePlanner

The SimplePlanner, as the name suggests, has little planning capability. The planner plans a straight line at a constant speed. The only logic of this planner is to decelerate if the current velocity exceeds the `max_velocity`.

Link to the code

6.2 IDMPlanner

The Intelligent Driver Model Planner (IDMPlanner) consists of two parts:

1. Path planning
2. Longitudinal control (IDM policy)

6.2.1 Path planning

The path planning is a breadth-first search algorithm. It finds a path towards the mission goal. The path consists of a serie of lane and lane connectors that leads to the roadblock containing the mission goal. The baseline is then extracted from the found path and is used as the reference path for the planner.

6.2.2 IDM Policy

Now that the planner has a reference path, it must then decide how fast to go along this path. For this, it follows the IDM policy. The policy describes how fast the planner should go based on the distance between itself and a given agent. Of course, it is wise to choose the closest agent in the path of the planner.

Hence, the IDMPlanner uses breadth-first search to find the path towards the mission goal, and the IDM policy describes how far along that path the planner should be.

Link to the code

6.3 UrbanDriverOpenLoopModel (MLPlanner)

The UrbanDriverOpenLoopModel functions as an example trained machine learning planner using the `MLPlanner` interface. The implementation is an open-loop version of L5Kit's implementation of "Urban Driver: Learning to Drive from Real-world Demonstrations Using Policy Gradients" adapted to the nuPlan framework.

The model processes vectorized agent and map inputs into local feature descriptors that are passed to a global attention mechanism for yielding a predicted ego trajectory. The model is trained using imitation learning to match expert trajectories available in the nuPlan dataset. Some amount of data augmentation is performed on the agents and expert trajectory provided during training to mitigate data distribution drift encountered during closed-loop simulation.

Link to the code

CHAPTER

SEVEN

METRICS

This document describes the available metrics in nuPlan to evaluate scenarios and score planners.

7.1 Common metrics in challenge 1:

In this challenge, expert driven trajectory and planner proposed trajectories are downsampled with a selected frequency (comparison_frequency = 1Hz). At each sampled time, the planner proposed trajectory and expert trajectory are compared up to the selected time horizon in the future (comparison_horizon = [3,5,8]s). The submission planners must meet the minimum required frequency of 1 Hz and the minimum planning horizon of 6s for this challenge.

Average Displacement Error (ADE) within bound: At each sampled time, ADE is defined as the average of pointwise L2 distances between the sampled planner proposed trajectory and expert trajectory starting at that time up to the selected comparison horizon in the future. ADE is calculated at all sampled times in the scenario. We define ADE score for a scenario based on the mean of all the calculated ADEs in that scenario.

Final Displacement Error (FDE) within bound: At each sampled time, FDE is defined as the L2 distance between the sampled planner proposed trajectory and expert trajectory at the final time available in the sampled trajectories (i.e., at the selected comparison horizon seconds in the future). FDE is calculated at all sampled times in the scenario. We define FDE score for a scenario based on the mean of all the calculated FDEs in that scenario.

Miss Rate within bound: At each sampled time, if the maximum of pointwise L2 distances between the sampled planner proposed trajectory and sampled expert trajectory up to the selected comparison horizon in the future is greater than its corresponding maximum displacement threshold, we consider the planner proposed trajectory at that time as a miss. This process is done for all sampled times in the scenario. Miss rate is defined as the ratio of sampled times where the corresponding trajectory was marked as a miss over the number of the sampled times. Nuboard visualizes miss rate ratio and a boolean metric (True if the miss rate is below a maximum acceptable threshold, max_miss_rate_threshold = 0.3, for all horizons in comparison_horizon = [3,5,8]s else False).

Average Heading Error (AHE) within bound: At each sampled time, average heading error is defined as the average of the absolute differences between the sampled planner proposed trajectory and expert trajectory starting at that time up to the selected comparison horizon in the future. Average heading error is calculated at all sampled times in the scenario. We define average heading error score for a scenario based on the mean of all the calculated average heading errors in that scenario.

Final Heading Error (FHE) within bound: At each sampled time, final heading error is defined as the abosulte difference between the sampled planner proposed trajectory and expert trajectory at the final time available in the sampled trajectories (i.e., at the selected comparison horizon seconds in the future). Final heading error is calculated at all sampled times in the scenario. We define final heading error score for a scenario based on the mean of all the calculated final heading errors in that scenario.

Metric	Thresholds	Visualization
Average Displacement Error (ADE) within bound	comparison_horizon = [3,5,8]s comparison_frequency = 1Hz max_average_l2_error_threshold = 8m	Histogram of: • mean of ADE (m)
Final Displacement Error (FDE) within bound	*comparison_horizon = [3,5,8]s *comparison_frequency = 1Hz max_final_l2_error_threshold = 8m	Histogram of: • mean of FDE (m)
Miss Rate within bound	*comparison_horizon = [3,5,8]s *comparison_frequency = 1Hz max_displacement_threshold = [6,8,16]m max_miss_rate_threshold = 0.3	Histograms of: • miss rate (ratio) • boolean (True if miss rate is below the max_miss_rate_threshold)
Average Heading Error (AHE) within bound	*comparison_horizon = [3,5,8]s *comparison_frequency = 1Hz max_average_heading_error_threshold = 0.8 rad	Histogram of: • mean of average heading errors (rad)
Final Heading Error (FHE) within bound	*comparison_horizon = [3,5,8]s *comparison_frequency = 1Hz max_final_heading_error_threshold = 0.8 rad	Histogram of: • mean of final heading errors (rad)

'*' : inherited threshold from the related lower level metric

7.2 Common metrics in challenge 2 and challenge 3:

In these challenges, expert and driven ego trajectories are compared to score a scenario.

No at-fault Collisions: A collision is defined as the event of ego's bounding box intersecting another agent's bounding box. If a collision lasts for multiple frames, it still counts as a single collision and the first frame is considered to calculate the transferred kinematic energy during the collision. All collided tracks will be removed from metrics evaluations at future frames after the collision.

We classify collisions into 5 groups: STOPPED_EGO_COLLISION (a collision that happens when ego is stopped), STOPPED_TRACK_COLLISION (a collision that happens when the track is stopped), ACTIVE_FRONT_COLLISION (a collision that happens when front bumper of active ego hits an active track), ACTIVE_REAR_COLLISION (a collision that happens when an active track hits ego in the rear) and ACTIVE_LATERAL_COLLISION (a collision that happens when an active track and ego hit on the sides). To define the collision score for a scenario, we only consider collisions that should have been prevented if planner performed properly. For simplicity, we call these collisions at-fault. STOPPED_TRACK_COLLISION, ACTIVE_FRONT_COLLISION, and ACTIVE_LATERAL_COLLISION when ego's footprint is not fully in a single lane or lane_connector (e.g. during a lane change) are considered to be at-fault. For at-fault collisions, kinematic energy of collision at the time of collision is calculated based on the loss of velocity during the collision, which represents the intensity of the collision of ego with the other agents. At fault collisions are separated based on the collided track types to 3 groups: Vulnerable Road Users (VRUs) including pedestrians and bicyclists, vehicles, and objects including all other predicted track types (traffic cones, generic objects etc.). See nuboard histogram tab for statistics on the number of at-fault collisions and min/max/mean of at-fault collisions energies (m/s) for each group. This metric contributes to the scenario score as a multiplier based on the number of at-fault collisions that happens in a scenario for each group and the acceptable thresholds (max_violation_threshold_vru = 0 max_violation_threshold_vehicle = 0, max_violation_threshold_object = 1).

Drivable area compliance: Ego should drive in the mapped drivable area at all times. Drivable area compliance metric identifies the frames when ego drives outside the drivable area (See related timeseries in the scenario tab in nuboard). Due to over-approximation of ego's bounding box, we allow for a small infringemenet outside the drivable area (max_violation_threshold = 0.3m). If there exists a frame where the maximum distance of the corners of ego's bounding box from the nearest drivable area is more than the selected threshold, drivable area compliance score is set to 0, otherwise it is set to 1. Nuboard histogram tab visualizes the boolean representation of this metric (True if there

is no drivable area violation more than max_violation_threshold else False). This boolean metric contributes to the scenario score as a multiplier.

Driving direction compliance: This is a metric defined to penalize ego when "it drives into oncoming traffic". The metric computes the movement of ego's center during a 1 second time_horizon along the driving direction defined according to the baselines of ego's lanes or lane-connectors. The score is set to 1 if it does not drive/move against the flow more than driving_direction_compliance_threshold (= 2 m) and 0 if it drives against the flow more than driving_direction_violation_threshold (= 6 m), and 0.5 otherwise.

Making progress: This metric is defined as a boolean metric based on the "Ego progress along the expert's route ratio" explained later in thos document. It's score is set to 1 if the ratio is more than the selected threshold (min_progress_threshold = 0.2), and is set to 0 otherwise. This boolean metric contributes to the scenario score as a multiplier.

Time to Collision (TTC) within bound: TTC is defined as the time required for ego and another track to collide if they continue at their present speed and heading. We only compute time to collision for tracks in front of ego and cross traffic, and lateral tracks on the sides only when ego's footprint is not fully in a single lane or lane_connector (e.g. during a lane change) or when ego is in areas marked as intersection where lanes may merge or extend into multiple lanes. To compute TTC, tracks and ego bounding boxes are projected with a selected time step (time_step_size = 0.1s) up to a selected time horizon in the future (time_horizon = 3.0s) and TTC is defined as the first time the projected bounding boxes intersect. If there is no intersection between the projected bounding boxes up to the selected time horizon, TTC is set to infinity. Nuboard histogram tab visualizes the min of TTC at all frames capped at time_horizon = 3.0s, and a boolean metric (True if TTC is higher than a minimum lower bound, least_min_ttc = 0.95s, else False). The boolean metric contributes to the scenario score in the weighted average function.

Ego progress along the expert's route ratio: We evaluate progress of the driven ego trajectory in a scenario by comparing its progress along the route that expert takes in that scenario. Expert's route is extracted as a sequence of lane and lane_connectors it moves along during the scenario. While ego is in the corresponding roadblocks of expert's route, progress per frame is computed based on progress along the baselines of expert's route. Sum/Integral of per frame ego's progress values (overall ego progress) and expert's progress values (overall expert progress) during the scenario are computated to define ego to expert progress ratio. Ego is not allowed to completely drive backwards in our scenarios. However, due to noise in data, in some scenarios where ego is stopped in the entire scenario the overall progress may take a small negative value up to - score_progress_threshold (score_progress_threshold = 0.1m). We set ego's to expert progress ratio to 0 if the overall ego progress is less than this negative threshold. The ratio is set to 1 in scenarios where expert is not assigned a route (i.e. is in car_park area, etc.). In other cases, the ratio is defined as the min(1 , max(overall ego progress, score_progress_threshold)/ max(overall expert progress, score_progress_threshold)). The ratio contributes to the scenario score in the weighted average function.

Speed limit compliance: This metric evaluates if ego's speed exceeds the associated speed limit in the map. The speed limit is queried from the lane ego is associated to. If ego is associated to a lane_connector, speed limit is set as the maximum of the speed limits of the incoming and outgoing lanes of that lane_connector. Speed limit violation at each frame is defined based on the difference between ego's speed and the speed limit, if ego's speed is higher than the speed limit (over-speeding). See nuboard histogram tab for statistics on the number of speed limit violations, a boolean metric (True if there is no speed limit violation else False), and min/max/mean of speed limit violations (m/s). Duration of the intervals when ego drives above the speed limit is considered in determining the mean value. This metric contributes to the scenario score in the weighted average function.

Comfort: We measure the comfort of ego's driven trajectory by evaluating minimum and maximum longitudinal accelerations, maximum absolute value of lateral acceleration, maximum absolute value of yaw rate, maximum absolute value of yaw acceleration, maximum absolute value of longitudinal component of jerk, and maximum magnitude of jerk vector. These variables are compared to thresholds with default values determined empirically from examination of a dataset of expert trajectories (min_lon_accel = -4.05 m/s^2, max_lon_accel = 2.40 m/s^2, max_abs_lat_accel = 4.89 m/s^2, max_abs_yaw_accel = 1.93 rad/s^2, max_abs_yaw_rate = 0.95 rad/s, max_abs_lon_jerk = 4.13 m/s^3, max_abs_mag_jerk = 8.37 m/s^3). Nuboard histogram tab visualizes a boolean metric ego_is_comfortable (True if all the mentioned variables are within the selected thresholds, else False). The boolean metric contributes to the scenario score in the weighted average function.

The following metrics are available in nuboard, but are not used in the scenario score.

Mean speed: This metric reports the average speed of trajectory during the scenario.

Displacement error: This metric computes the pointwise L2 distances between the ego driven trajectory and expert at all timepoints in the scenario. A discount factor (discount_factor = 1) is added to reduce the effect of later timepoints in the scenario if needed. Nuboard visualizes timeseries of the error and histogram of max/min/mean values.

Displacement error with yaw: This metric computes the pointwise L2 distances between the ego driven trajectory and expert with a weighted sum of the absolute difference in their headings (heading_diff_weight = 2.5) at all timepoints in the scenario. A discount factor (discount_factor = 1) is added to reduce the effect of later timepoints in the scenario if desired. Nuboard visualizes the error and histogram of max/min/mean values.

Metric	Thresholds	Visualization
No at-fault collision	max_violation_threshold_vru = 0 max_violation_threshold_vehicle = 0 max_violation_threshold_object = 1	Histograms of: • number of at-fault collisions • min/max/mean of at-fault collisions energy, if exists (m/s)
Drivable Area Compliance	max_violation_threshold = 0.3 m	Histogram of: • boolean (True if there is no drivable area violation more than max_violation_threshold else False)
Driving Direction Compliance	driving_direction_violation_threshold = 2 m driving_direction_violation_threshold = 6 m time_horizon = 1 s	Histogram of: • score value
Making progress	min_progress_threshold = 0.2	Histogram of: • boolean (True if progress ratio is more than min_progress_threshold else False)
Time to Collision (TTC) within bound	time_step_size = 0.1s time_horizon = 3.0s least_min_ttc = 0.95s	Histogram of: • min of TTC (s) • boolean (True if TTC is higher than least_min_ttc, else False)
Speed limit compliance	max_compliance_threshold = 1.0 max_overspeed_value_threshold = 2.23 m/s (5mph)	Histogram of: • number of speed limit violations • boolean (True if there is no speed limit violation else False)
Ego progress along the expert's route ratio	score_progress_threshold = 0.1 m	Histogram of: • mean of per frame ego's progress values over the scenario (m) • Sum/Integral of per frame ego's progress values (Overall progress) over the scenario (m) • Ratio of ego's progress along the expert route to a desired progress threshold, default set at expert's overall progress. The ratio is saturated at 0 and 1.
Comfort	min_lon_accel = -4.05 m/s^2 max_lon_accel = 2.40 m/s^2 max_abs_lat_accel = 4.89 m/s^2 max_abs_yaw_accel = 1.93 rad/s^2 max_abs_yaw_rate = 0.95 rad/s max_abs_lon_jerk = 4.13 m/s^3 max_abs_mag_jerk = 8.37 m/s^3	Histogram of: • boolean (True if all metrics are within the acceptable thresholds else False)
Displacement error	discount_factor = 1	Histogram of: • min/max/mean of errors
Displacement error with yaw	discount_factor = 1 heading_diff_weight = 2.5	Histogram of: • min/max/mean of errors

CHAPTER

EIGHT

FINAL METRIC STRUCTURE

This document describes how metrics are aggregated to generate the final score structure for comparison of planners performance on AV in Nuplan.

8.1 Individual metric scores

We use a different set of metrics for evaluation of planners in challenge 1 and challenges 2 and 3. Each metric in the scenario score is assigned a value/score between 0-1 based on the planner performance in that scenario. A higher score shows a better performance according to that metric (See metrics scores' description in the table below and a more comprehensive description about metrics in general in metric_description.

8.2 Planner score in each scenario based on the proposed trajectory in challenge 1

In each scenario, selected metrics are aggregated to provide a score for the proposed trajectory. Currently, the aggregation function is a hybrid hierarchical-weighted average function of individual metric scores:

- The planner gets a zero score in a scenario if the miss rate is above the selected threshold,
- Otherwise, a weighted average of other metrics' scores is used as the score in that scenario.

Metric Name	Metric Score	Weight in Scenario Score
Miss Rate Within Bound	0 if maximum displacement error between the planner proposed trajectory and expert trajectory is more than the `max_displacement_threshold` at the corresponding `comparison_horizon` in more than `max_miss_rate_threshold` of the time instances in the scenario, 1 otherwise.	NA/multiplying metric
Average Displacement Error (ADE) Within Bound	0 if the average of ADEs over the `comparison_horizon` is more than `max_average_l2_error_threshold`, 1 otherwise.	1
Final Displacement Error (FDE) Within Bound	0 if the average of ADEs over the `comparison_horizon` is more than `max_final_l2_error_threshold`, 1 otherwise.	1
Average heading error (AHE) Within Bound	0 if the average of AHEs over the `comparison_horizon` is more than `max_average_heading_error_threshold`, 1 otherwise.	2
Final heading error (FHE) Within Bound	0 if the average of FHEs over the `comparison_horizon` is more than `max_final_heading_error_threshold`, 1 otherwise.	2

8.3 Planner score in each scenario based on the driven trajectory in challenges 2 and 3

In each scenario, selected metrics are aggregated to provide a score for the driven trajectory. Currently, the aggregation function is a hybrid hierarchical-weighted average function of individual metric scores:

- The planner gets a zero score for that driven trajectory/scenario if
  - there is an at_fault collision with a vehicle or a VRU (pedestrian or bicyclist), or
  - there are multiple at_fault collisions with objects (e.g. a cone), or
  - there is a drivable_area violation,
  - ego drives into uncoming traffic more than 6 m, or
  - ego is not making enough progress.
- A weighted average of other metrics' scores is multiplied with 0.5 if there is one at_fault collision with an object (e.g. a cone), or if ego drives into uncoming traffic more than 2 m (but less than 6 m).
- Otherwise, a weighted average of other metrics' scores is used as the score in that scenario.

Metrics scores, and how they are aggregated to compute the scenarios score is described in the following table. You can find metric thresholds/constants in metric_description.

Metric Name	Metric Score	Weight in Scenario Score
no_ego_at_fault_collisions	0 if there is an at-fault collision with a vehicle or a vru, or multiple at-fault collisions with objects, 0.5 if there's an at-fault collision with a single object, 1 otherwise.	NA/multiplying metric
drivable_area_compliance	0 if at any instance the distance of a corner of ego's bounding box from the drivable area is more than `max_violation_threshold`, 1 otherwise.	NA/multiplying metric
driving_direction_compliance	Score is 1 if during the previous `time_horizon` (for each time instance) ego has not been driving against the traffic flow more than `driving_direction_compliance_threshold`, and 0 if it's been driving against the flow more than `driving_direction_violation_threshold`, and 0.5 otherwise.	5
time_to_collision_within_bound	0 if time_to_collision is less than `least_min_ttc` threshold, 1 otherwise.	5
speed_limit_compliance	Score is `max(0, 1 - (speed_violation_integral / (max_overspeed_value_threshold * total_scenario_duration)))` where `speed_violation_integral` is area under the speed_violation vs time graph, `max_overspeed_value_threshold` is the maximum acceptable over-speeding threshold, currently set at 2.23 m/s (equivalent to ~5mph), and `total_scenario_duration` is the scenario duration in seconds. Therefore, score is 1 if there is no speed limit violation and approaches to 0 as the violation increases.	4
ego_progress_along_expert_route	Score is 0 if `overall_ego_progress<0`, otherwise it is `min(1.0, max(overall_ego_progress, score_progress_threshold)/ max(overall_expert_progress, score_progress_threshold)` where `overall_ego_progress` is ego's overall progress along the expert route, and `overall_expert_progress` is expert overall progress along its route, and `score_progress_threshold` is a small threshold.	5
ego_is_making_progress	0 if `ego_progress_along_expert_route` returns a value less than `min_progress_threshold`, 1 otherwise.	NA/multiplying metric
Ego_is_comfortable • ego_jerk • ego_lat_acceleration • ego_lon_acceleration • ego_lon_jerk • ego_yaw_acceleration • Ego_yaw_rate	0 if any of the comfort_metrics are not within comfort bounds/thresholds and 1 otherwise.	2

8.4 Planner score in each scenario type

nuPlan categorizes scenarios based on their types. The type is specified based on features of the scenario according to ego, agents, and the scene. A planner's score on one specific scenario type can be computed by averaging the scores of driven trajectories that belong to that scenario type. This score can be helpful in identifying the scenario types in which the planner does not perform well. This can be particularly helpful for ML planners. For instance if a planner gets an average score of 0.9 for 50 scenarios with scenario_type ON_INTERSECTION, and an average score of 0.6 for 60 scenarios of type ON_PICKUP_DROPOFF, we may want to consider improving the performance in scenarios where ego is on PUDO area.

8.5 Planner final score

A planner is assigned a score between 0 and 1 based on its performance in multiple scenarios. A higher score shows a better performance. To assign a final score to a planner, it is run on N scenarios, and the driven trajectory scores in these N scenarios are averaged to score the planner.

CHAPTER

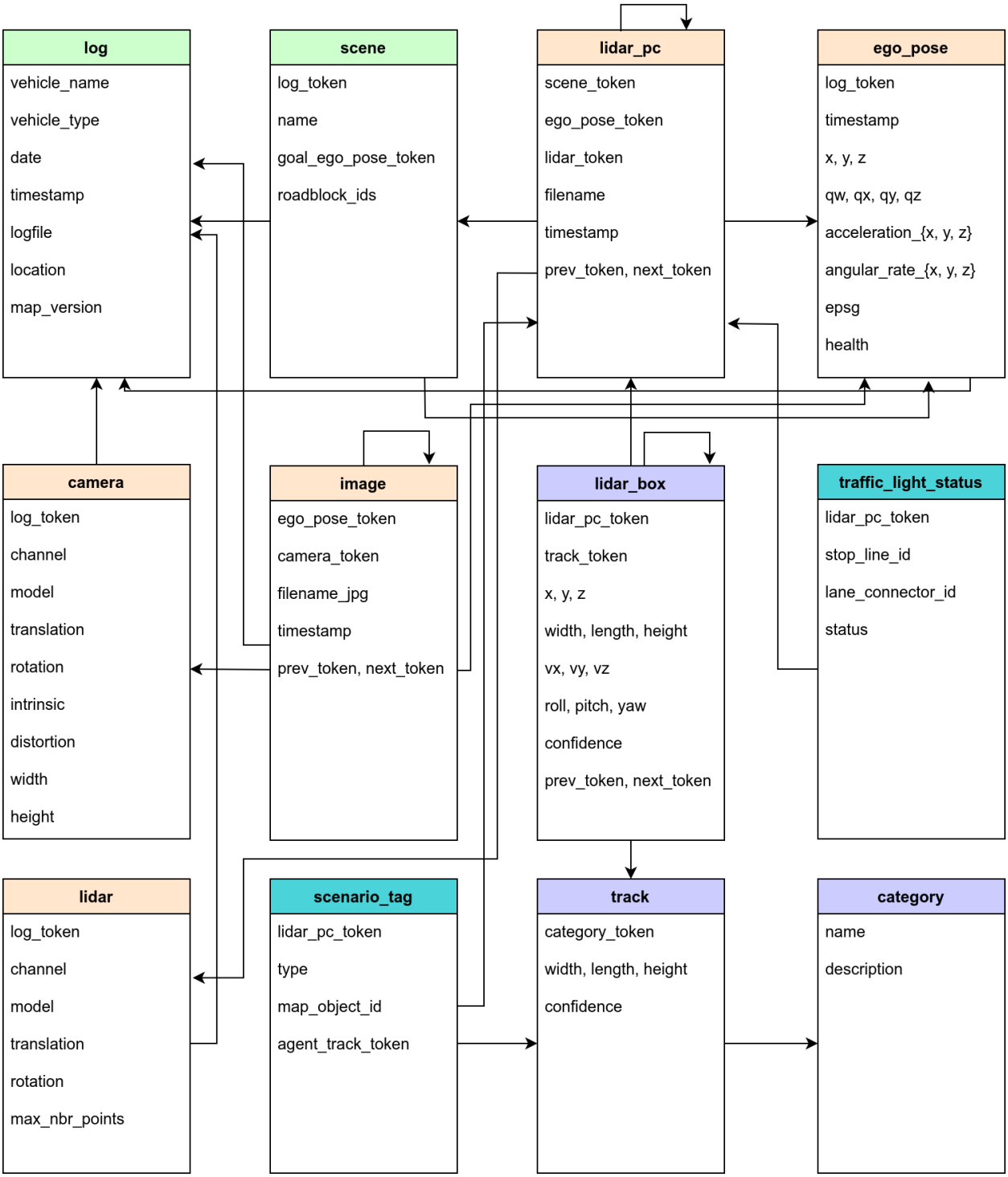
NINE

NUPLAN SCHEMA

This document describes the database schema used in nuPlan. All annotations and metadata (including calibration, maps, vehicle coordinates etc.) are covered in relational databases. The database tables are listed below. Every row can be identified by its unique primary key `token`. Foreign keys such as `log_token` are used to link to the `token` of the table `log`.

nuPlan schema

Every table has a unique identifier called `token`. Foreign keys are indicated as `<table>_token`.
We only show fields that are stored in the DB, not additional information that may be retrieved via convenience functions.
Identifiers for map objects are indicated as `<map_layer>_id(s)`.



9.1 log

Information about the log from which the data was extracted.

```
log {
   "token":                        <str> -- Unique record identifier.
   "vehicle_name":                 <str> -- Vehicle name.
   "date":                         <str> -- Date (YYYY-MM-DD).
   "timestamp":                    <int> -- Unix timestamp for when the log started.
   "logfile":                      <str> -- Original log file name.
   "location":                     <str> -- Area where log was captured, e.g. Singapore.
   "map_version":                  <str> -- Name of map version used in this log.
}
```

9.2 ego_pose

Ego vehicle pose at a particular timestamp.

```
ego_pose {
   "token":                        <str> -- Unique record identifier.
   "log_token":                    <str> -- Foreign key. Identifies the log which this␣
↪ego pose is a part of.
   "timestamp":                    <str> -- Unix timestamp.
   "x":                            <float> -- Ego vehicle location center_x in global␣
↪coordinates(in meters).
   "y":                            <float> -- Ego vehicle location center_y in global␣
↪coordinates(in meters).
   "z":                            <float> -- Ego vehicle location center_z in global␣
↪coordinates(in meters).
   "qw":                           <float> -- Ego vehicle orientation in quaternions in␣
↪global coordinates.
   "qx":                           <float> -- Ego vehicle orientation in quaternions in␣
↪global coordinates.
   "qy":                           <float> -- Ego vehicle orientation in quaternions in␣
↪global coordinates.
   "qz":                           <float> -- Ego vehicle orientation in quaternions in␣
↪global coordinates.
   "vx":                           <float> -- Ego vehicle velocity x in local␣
↪coordinates(in m/s).
   "vy":                           <float> -- Ego vehicle velocity y in local␣
↪coordinates(in m/s).
   "vz":                           <float> -- Ego vehicle velocity z in local␣
↪coordinate(in m/s)s.
   "acceleration_x":               <float> -- Ego vehicle acceleration x in local␣
↪coordinates(in m/s2).
   "acceleration_y":               <float> -- Ego vehicle acceleration y in local␣
↪coordinates(in m/s2).
   "acceleration_z":               <float> -- Ego vehicle acceleration z in local␣
↪coordinates(in m/s2).
   "angular_rate_x":               <float> -- Ego vehicle angular rate x in local␣
↪coordinates.
```

(continues on next page)

(continued from previous page)

```
   "angular_rate_y":               <float> -- Ego vehicle angular rate y in local␣
↪coordinates.
   "angular_rate_z":               <float> -- Ego vehicle angular rate z in local␣
↪coordinates.
   "epsg":                         <int> -- Ego vehicle epsg. Epsg is a latitude/
↪longitude coordinate system based on the Earth's center of mass.
}
```

9.3 camera

The camera table contains information about the calibration and other settings of a particular camera in a particular log.

```
camera {
   "token":                        <str> -- Unique record identifier.
   "log_token":                    <str> -- Foreign key. Identifies the log that uses␣
↪the configuration specified here.
   "channel":                      <str> -- The camera name, which describes it's␣
↪position on the car (e.g. 'CAM_F0', 'CAM_R0', 'CAM_R1', 'CAM_R2', 'CAM_B0', 'CAM_L0',
↪'CAM_L1', 'CAM_L2').
   "model":                        <str> -- The camera model used.
   "translation":                  <float> [3] -- The extrinsic translation of the␣
↪camera relative to the ego vehicle coordinate frame. Coordinate system origin in␣
↪meters: x, y, z.
   "rotation":                     <float> [4] -- The extrinsic rotation of the camera␣
↪relative to the ego vehicle coordinate frame. Coordinate system orientation as␣
↪quaternion: w, x, y, z.
   "intrinsic":                    <float> [3, 3] -- Intrinsic camera calibration matrix.
   "distortion":                   <float> [*] -- The camera distortion parameters␣
↪according to the Caltech model (k1, k2, p1, p2, k3)
   "width":                        <int> -- The width of the camera image in pixels.
   "height":                       <int> -- The height of the camera image in pixels.
}
```

9.4 image

The image table stores metadata to retrieve a single image taken from a camera. It does not store the image itself.

```
image {
   "token":                        <str> -- Unique record identifier.
   "next_token":                   <str> -- Foreign key. Record that follows this in␣
↪time. Empty if last image of log.
   "prev_token":                   <str> -- Foreign key. Record that precedes this in␣
↪time. Empty if first image of log.
   "ego_pose_token":               <str> -- Foreign key. Indicates the ego pose at the␣
↪time that the image was captured.
   "camera_token":                 <str> -- Foreign key. Indicates the camera settings␣
↪used to capture the image.
```

(continues on next page)

(continued from previous page)

```
   "filename_jpg":                <str> -- Relative path to image file.
   "timestamp":                   <int> -- Unix timestamp.
}
```

9.5 lidar

The `lidar` table contains information about the calibration and other settings of a particular lidar in a particular log.

```
lidar {
   "token":                       <str> -- Unique record identifier.
   "log_token":                   <str> -- Foreign key. Identifies the log that uses␣
↪the configuration specified here.
   "channel":                     <str> -- Log channel name.
   "model":                       <str> -- The lidar model.
   "translation":                 <float> [3] -- The extrinsic translation of the lidar␣
↪relative to the ego vehicle coordinate frame. Coordinate system origin in meters: x, y,
↪ z.
   "rotation":                    <float> [4] -- The extrinsic rotation of the lidar␣
↪relative to the ego vehicle coordinate frame. Coordinate system orientation as␣
↪quaternion: w, x, y, z.
}
```

9.6 lidar_pc

The `lidar_pc` table stores metadata to retrieve a single pointcloud taken from a lidar. It does not store the pointcloud itself. The pointcloud combines sweeps from multiple different lidars that are aggregated on the car.

```
lidar_pc {
   "token":                       <str> -- Unique record identifier.
   "next_token":                  <str> -- Foreign key. Record that follows this in␣
↪time. Empty if end of log.
   "prev_token":                  <str> -- Foreign key. Record that precedes this in␣
↪time. Empty if start of log.
   "scene_token":                 <str> -- Foreign key. References the scene that␣
↪contains this lidar_pc.
   "ego_pose_token":              <str> -- Foreign key. Indicates the ego pose at the␣
↪time that the pointcloud was captured.
   "lidar_token":                 <str> -- Foreign key. Indicates the lidar settings␣
↪used to capture the pointcloud.
   "filename":                    <str> -- Relative path to pointcloud blob.
   "timestamp":                   <int> -- Unix timestamp.
}
```

9.7 lidar_box

The `lidar_box` table stores individual object annotations in the form of 3d bounding boxes. These boxes are extracted from an Offline Perception system and tracked across time using the `track` table. Since the perception system uses lidar as an input modality, all `lidar_boxes` are linked to the `lidar_pc` table.

```
lidar_box {
   "token":                        <str> -- Unique record identifier.
   "lidar_pc_token":               <str> -- Foreign key. References the lidar_pc where␣
↪this object box was detected.
   "track_token":                  <str> -- Foreign key. References the object track␣
↪that this box was associated with.
   "next_token":                   <str> -- Foreign key. Record that follows this in␣
↪time. Empty if end of track.
   "prev_token":                   <str> -- Foreign key. Record that precedes this in␣
↪time. Empty if start of track.
   "x":                            <float> -- Bounding box location center_x in global␣
↪coordinates(in meters).
   "y":                            <float> -- Bounding box location center_y in global␣
↪coordinates(in meters).
   "z":                            <float> -- Bounding box location center_z in global␣
↪coordinates(in meters).
   "width":                        <float> -- Bounding box size width(in meters).
   "length":                       <float> -- Bounding box size length(in meters).
   "height":                       <float> -- Bounding box size height(in meters).
   "vx":                           <float> -- Bounding box velocity v_x(in m/s). This␣
↪quantity is estimated by a Neural Network and may be inconsistent with future␣
↪positions.
   "vy":                           <float> -- Bounding box velocity v_y(in m/s). This␣
↪quantity is estimated by a Neural Network and may be inconsistent with future␣
↪positions.
   "vz":                           <float> -- Bounding box velocity v_z(in m/s). This␣
↪quantity is estimated by a Neural Network and may be inconsistent with future␣
↪positions.
   "yaw":                          <float> -- Bounding box orientation yaw.
   "confidence":                   <float> -- Bounding box confidence.
}
```

9.8 track

An object track, e.g. particular vehicle. This table is an enumeration of all unique object instances we observed.

```
track {
   "token":                        <str> -- Unique record identifier.
   "category_token":               <str> -- Foreign key. Object instance category.
   "width":                        <float> -- Bounding box size width(in meters).
   "length":                       <float> -- Bounding box size length(in meters).
   "height":                       <float> -- Bounding box size height(in meters).
}
```

9.9 category

Taxonomy of object categories (e.g. 'vehicle', 'bicycle', 'pedestrian', 'traffic_cone', 'barrier', 'czone_sign', 'generic_object').

```
category {
   "token":                   <str> -- Unique record identifier.
   "name":                    <str> -- Category name.
   "description":             <str> -- Category description.
}
```

9.10 scene

A `scene` is a snippet of upto 20s duration from a `log`. Every scene stores a goal for the ego vehicle that is a future ego pose from beyond that scene. In addition we provide a sequence of road blocks to navigate towards the goal.

```
scene {
   "token":                   <str> -- Unique record identifier.
   "log_token":               <str> -- Foreign key. The log that this scene is a␣
↪part of.
   "name":                    <str> -- Unique Scene Name.
   "goal_ego_pose_token":     <str> -- Foreign key. A future ego pose that serves␣
↪as the goal for this scene.
   "roadblock_ids":           <str> -- A sequence of roadblock ids separated by␣
↪commas. The ids can be looked up in the Map API.
}
```

9.11 scenario_tag

An instance of a scenario extracted from the database. Scenarios are linked to `lidar_pcs` and represent the point in time when a scenario miner was triggered, e.g. when simultaneously being in two lanes for `CHANGING_LANE`. Some scenario types optionally can refer to an agent that the ego is interacting with (e.g. in `STOPPING_WITH_LEAD`).

```
scenario_tag {
   "token":                   <str> -- Unique record identifier.
   "lidar_pc_token":          <str> -- Foreign key. The lidar_pc at which this␣
↪scenario was triggered.
   "type":                    <str> -- Type of Scenario. Ex on_intersection,␣
↪starting_unprotected_cross_turn etc. There are around 70 of these scenario types.
   "agent_track_token":       <str> -- Foreign key. Token of the agent interacting␣
↪with the ego vehicle. Can be none if there is no interacting agent.
}
```

9.12 traffic_light_status

We use the observed motion of agents in the environment to estimate the status of a traffic light. For simplicity the status is stored in a particular lane connector (an idealized path across an intersection), rather than in the traffic light itself.

```
traffic_light_status {
   "token":                        <str> -- Unique record identifier.
   "lidar_pc_token":               <str> -- Foreign key. The traffic light status is␣
↪based on the motion of the agents detected in this lidar_pc.
   "lane_connector_id":            <int> -- ID of lane connector in the map.
   "status":                       <str> -- Status of traffic light. Can be green, red␣
↪or unknown.
}
```

CHAPTER

TEN

FAQ

10.1 Q: Preprocessing (cache) takes forever

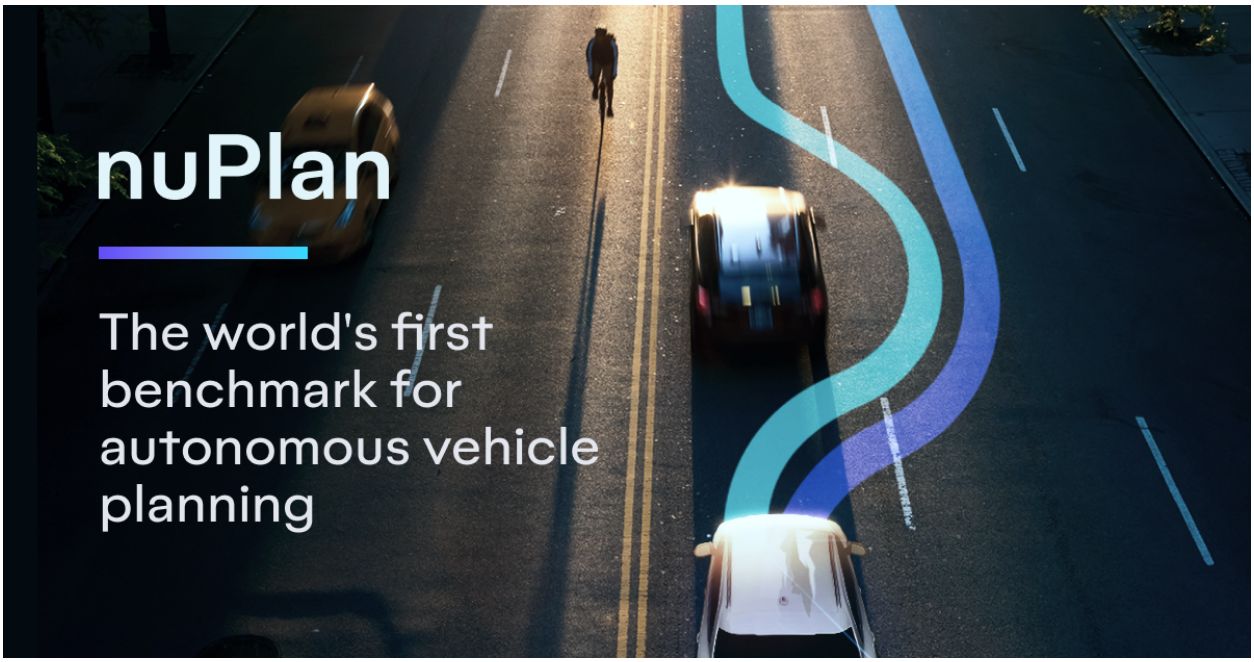
A: **Preprocessing could be bottlenecked by file I/O**. Make sure your dataset is placed at a location with enough I/O bandwith, eg. *local SSDs*. Since our feature builder queries the database files, lacking of I/O bandwith bottlenecks the query speed. Also please check the amount of samples you want to preprocess (details in the next point).

10.2 Q: How many samples should I use? / Dataset size / Subsampling

A: **Preprocessing the full dataset exaustively may not be ideal**. By default we use `NuPlanScenario` so that it generates a sample per timestep in the database, which is `20Hz` – very dense. By setting `scenario_filter.limit_total_scenarios=0.1` for example, it reduces to `2Hz` so that the number of samples by `1/10`. Note that this rate could be independent to the frequency used in input/target features. You can always have your desired sampling rate of input/target features in your customized `FeatureBuilder` or `TargetBuilder` when extracting features.

10.3 Q: My gradient becomes NaN / Numerical stability / Float precision

A: **FP16 causes numerical instability problems**. Before deep diving into your own model, make sure your `lightning.trainer.params.precision` is set to `32`. In the default configuration we set `lightning.trainer.params.precision=16`, meaning that FP16 is used for training. If you are not 100% sure about your model is numerically stable using FP16, please always use FP32.


nuPlan
The world's first benchmark for autonomous vehicle planning