Practical Course Report: nuPlan Planning Challenge

Yiderigun

***Abstract*—Autnomous driving vehicles are expected to deliver substantial reductions in accidents and traffic congestion, along with economic and environmental benefits. Nonetheless, there are several challenges to overcome before realizing this utopian scenarios. A critical aspect of the self-driving system, currently acting as a bottleneck impeding the progress of autonomous driving, is the planning module. This study addresses this challenge by employing an offline learning policy gradient approach, using a bird's-eye view (BEV) as the input for imitative policy learning.**

***Index Terms*—Autonomous Driving, Machine Learning, Bird's -Eye View, Graph Neural Network, Imitation Learning**

I. Introduction

THE planning module stands as a pivotal element within the self-driving system, yet it serves as a major obstacle impeding the advancement of autonomous driving. Its core purpose revolves around deciding the most suitable action for the self-driving vehicle under any given circumstance. In a traditional rule-based planning approach, engineers employ a manually crafted loss function to choose a trajectory that minimizes errors. Nevertheless, enhancing its effectiveness demands engineers to create fresh terms in the loss function or fine-tune the weights of existing ones for every driving scenario. This procedure is not only financially burdensome but also inefficient when expanding to new geographic locations.

The research conducted by Bansal [1] has demonstrated the early adoption of machine-learning-based policies for autonomous driving, acquired directly from human demonstrations. While these approaches offer improved scalability compared to the traditional hand-engineered method, they fall short in terms of providing the essential interpretability and safety assurances required for the secure deployment of such systems in a production environment.

Recent breakthroughs in rich mid-level representations, driven by extensive datasets [2][3], high-definition maps, and high-performance perception systems, have enabled the capture of the intricacies of urban driving. This advancement has paved the way for the development of innovative techniques that achieve impressive results in motion prediction [4]. Furthermore, it has been shown that by leveraging these representations and employing behavioral cloning with state perturbations, it is possible to acquire robust driving policies. However, a significant challenge in this approach is the engineering of the perturbation noise mechanism necessary to address covariate shift between the training and testing distributions.

Taking inspiration from this approach, we present the preliminary results of our offline learning strategy, which centers on emulating driving policies using mid-level representations and a policy gradient technique. This framework offers several benefits: it can proficiently acquire intricate maneuvers without requiring perturbations, inherently tackles the covariate shift challenge, and directly optimizes both imitation and auxiliary costs. Our proposed simulator is constructed directly from the recorded logs of real-world driving demonstrations and high-definition maps of the targeted region, enabling the generation of new, lifelike driving scenarios based on past experiences. Moreover, when dealing with extensive datasets, the reduction of computational complexity becomes paramount. We harness vectorized representations and illustrate how they enable the efficient computation of policy gradients over time using backpropagation.

We utilize the dataset and path planning simulator derived from nuPlan [5]. nuPlan represents the world's first comprehensive planning benchmark designed for autonomous driving. While machine learning-based motion planners are gaining momentum, progress in this field has been hindered due to the absence of well-established datasets, simulation frameworks, and evaluation metrics. Existing benchmarks such as Argoverse, Lyft, and Waymo primarily focus on short-term motion prediction of other vehicles, rather than addressing the long-term planning aspect for the ego vehicle. This limitation has led previous research to rely on open-loop evaluation using L2-based metrics, which are not suitable for a fair assessment of long-term planning capabilities. Our benchmark addresses these challenges by offering a training framework for the development of machine learning-based planners, a streamlined closed-loop simulator, metrics tailored to motion planning, and an interactive visualization tool to present the results effectively.

In the following sections, we will introduce related work, methodology that is utilized, results and evaluation, introduction fo nuPlan datasets and Docker,which is utilized to create a valid submsission in the nuPlan Planning challenge.

II. Related Work

Within this section, we provide a concise overview of various methods employed for addressing decision-making challenges in the context of self-driving vehicles, drawing from both academic research and industry applications. We place specific emphasis on two broad categories: optimization-based systems and machine learning-based systems.

A. Optimization-based System

Typically, these systems tackle the issue by framing it as an optimization challenge, involving the manual design of a cost function. The objective is to produce an optimal trajectory by minimizing this cost function, a task accomplished using a range of optimization techniques like A* search-based methods [6], sampling-based approaches, dynamic programming, or combinations thereof that decompose the problem hierarchically [7].

Nevertheless, crafting an objective function that effectively manages comfort, safety, and route advancement across a diverse driving situations proves to be a challenging task [8]. Conversely, the inclusion of rigid constraints like obstacle avoidance and adherence to physical feasibility demands significantly less engineering input. As a result, creating a simplified, manually engineered system capable of recognizing and rectifying impractical trajectories represents a more straightforward and scalable approach.

Furthermore, these hand-crafted methods do not improve their performance with the addition of data, and their efficacy does not extend to intricate urban settings characterized by a high level of disorder. Consequently, substantial engineering efforts are imperative to fine-tune them, especially when expanding into new operational design domains or diverse geographic areas.

B. Machine-learning-based System

The major benefit of this approach lies in its ability to eliminate the requirement for manually crafted rules and its capacity to scale effectively with data, resulting in continuous performance enhancement as more training data becomes available [9]. Consequently, it holds great promise for addressing a wide range of driving scenarios. In the upcoming sections, we will delve into the two most commonly utilized machine learning paradigms for motion planning.

1) *Imitation Learning(IL)*
Imitation Learning is a form of supervised learning in which a model acquires expertise by mimicking expert behavior. The origins of IL in autonomous driving can be traced back to ALVINN[10], a system developed in 1989 that translated sensor data into steering commands, enabling autonomous navigation on rural roads. Although recent studies have showcased end-to-end driving using multiple-camera inputs alone, their practical results in real-world driving scenarios have been limited to straightforward tasks like lane following or navigating urban environments with low traffic density. To address the covariate shift challenge in IL, ChauffeurNet introduced the concept of applying IL to a bird's eye view(BEV) of the scene and incorporating synthetic perturbations [11].

2) *Reinforcement Learning(RL) & Inverse Reinforcement Learning(IRL)*
Reinforcement Learning (RL) represents a branch of machine learning particularly well-suited for handling sequential decision-making processes, a characteristic highly relevant to domains like self-driving, where agents must interact with their environment. In the context of autonomous driving, several approaches have been proposed, as documented in [12]. For example, [12] suggests a strategy that combines learned and rule-based components, which shares similarities with our own approach. However, it's worth noting that the results reported for this method are solely based on simulations. Another prevalent machine learning paradigm employed in autonomous driving is Inverse Reinforcement Learning (IRL) [13], which deduces the reward function from expert demonstrations and a model of the environment. Nonetheless, the effectiveness of all these techniques in real-world urban driving scenarios remains unassessed.

III. Method

In this section, detailed description on the practical implementation of path planning algorithm.

A. Differentiable Traffic Simulator from Real-world Driving Data

Within this subsection, we present a differentiable simulator denoted as S. This simulator serves as an approximation of driving situations, derived from a real-world experience denoted as $\bar{\tau}$ that has been collected. During the policy learning process, this simulator plays a pivotal role by allowing us to evaluate the performance of the current policy and facilitating the computation of the policy gradient. The utilization of this differentiable simulator enables us to assess how the current policy influences simulated driving scenarios effectively. This, in turn, allows for iterative enhancements and optimization of the policy through gradient-based methods.

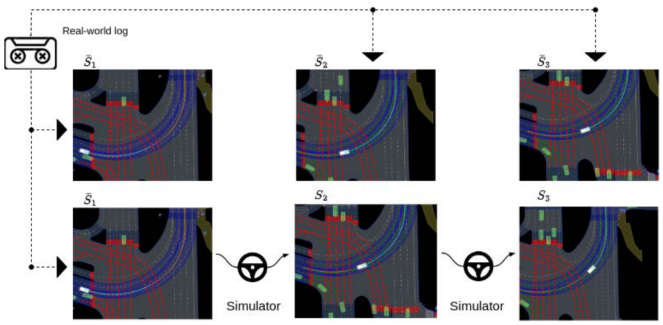

Fig. 1. Inspired by [14], differentiable simulator from observed real-world logs: based on a ground truth log (top row), model predicts a new trajectory corresponding to different SDV actions(bottom row)

We represent the real-world experience $\bar{\tau}$ as a sequence of state observation $\bar{s_t}$ around the vehicle over time:

$$\bar{\tau} = \{\bar{s_1}, \bar{s_2}, ..., \bar{s_T}\} \tag{1}$$

Each state observation $\bar{s_t}$ is composed of static and dynamic information. The static elements encompass divers entries such as traffic lanes, stop signs, pedestrians crossings. The dynamic elements consists of real-time information such as traffic lights, location of other traffic participants.

The updated poses of SDVs are determined by a kinematic model $p_{t+1} = f(p_t, a_t)$, p_t is trajectory of SDVs and a_t is action of drivers. We assume that kinematic model is differentiable. Furthermore, we can compute the gradients with respect to the state(S_s) and the action(S_a).

B. Vectorized model

Drawing inspiration from VectorNet[3], our approach centers on acquiring a unified representation that directly captures both the dynamics of multiple agents and the structured

context of the scene from their vectorized form, as depicted in Figure 2. on the right. Within this vectorized representation, the road features exhibit a geographical extent and can be expressed as points, polygons, or curves in geographic coordinates. For example, a lane boundary comprises multiple control points that collectively form a spline, a crosswalk is outlined by several points that shape a polygon, and a stop sign is represented by a single point. These geographical entities can be effectively approximated as polylines, which are delineated by multiple control points, each accompanied by its relevant attributes.

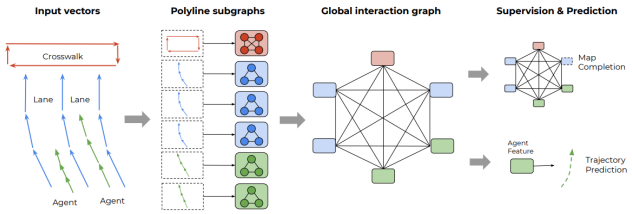

Fig. 2. An overview of VectorNet. Observed agent trajectories and map features are represented as sequence of vectors, and passed to a local graph network to obtain polyline-level features. Such features are then passed to a fully-connected graph to model the higher-order interactions. [3]

The behavior of moving agents can also be estimated by converting their motion trajectories into polylines. By expressing these polylines as collections of vectors, we can encompass both the spatial and temporal dimensions of the agents' movements and merge them with the structured context of the scene. This vectorized representation provides a consolidated and thorough comprehension of both the multi-agent dynamics and the scene's context.

We utilize Graph Neural Networks (GNNs) to incorporate these sets of vectors. In our methodology, every vector is regarded as a node within the graph, and the node characteristics are determined by factors such as the starting location, ending location, and additional attributes associated with each vector. These attributes could encompass information like the polyline group ID and semantic labels, among other possibilities. Through the creation of a graph-based representation, we can adeptly capture the contextual details derived from the high-definition maps and the paths of other mobile agents.

We introduce a hierarchical graph architecture to ensure that graph connectivity is based on the spatial and semantic proximity of the nodes. In this architecture, vectors that belong to the same polylines with identical semantic labels are connected and embedded into polyline features. Additionally, all polylines are fully connected to each other to facilitate the exchange of information.

C. Representing Trajectories and Maps

We utilize a sequence-based method to approximate the elements(such as lanes, intersection region and traffic lights) in a vectorized form. For map features, we select a starting point and direction, then sample key points from the splines uniformly. Then connect the neighbouring key points to construct vectors. Similarly, we sample key points at fixed intervals starting from $t = 0$, and connect them to construct vectors.

Inspired by VectorNet[3], each vector v_i, belonging to a polyline P_j is treated as a node in the graph. By leveraging graph neural networks, we can effectively process and analyze the information encoded in the graph representation. The graph structure allows for capturing the relationships and interactions between different vectors within the polyline sets, facilitating a comprehensive understanding of the spatial and semantic context of the trajectories and map annotations.

$$v_i = [d_i^s, d_i^e, a_i, j] \tag{2}$$

In the equation, d_i^s, d_i^e represent the coordinates of the start and end points of the vector. The variable a_i corresponds to attribute features, which can include object type, timestamps for trajectories, or road feature type and speed limit for lanes. The variable j is the ID of polyline P_j.

We achieve coordinate normalization for all vectors by centering them around the position of the target agent at its most recent observed time step. This normalization procedure guarantees that the input node characteristics remain unchanged regardless of the precise positions of the target agents.

D. Constructing the Polyline Subgraphs

For a given polyline P with its associated nodes $v_1, v_2, ..., v_p$, we define a single layer of subgraph propagation operation as:

$$v_i^{l+1} = \phi_{rel}(genc(v_i^l), \phi_{agg}(genc(v_j^l))) \tag{3}$$

In this equation, v_i^l represents the node features for the l-th layer of the subgraph network, and v_i^0 corresponds to the input features v_i. The function $genc(\bullet)$ transforms the individual node features, $\phi_{agg}(\bullet)$ aggregates the information from all neighbouring nodes, and $\phi_{rel}(\bullet)$ represents the relational operator between the node v_i and its neighbors.

In practice, the function $genc(\bullet)$ is implemented as a multi-layer perceptron (MLP) with shared weights across all nodes. The MLP consists of a single fully connected layer, followed by layer normalization and ReLU non-linearity. The function $agg(\bullet)$ performs maxpooling operation, while $rel(\bullet)$ simply concatenates the features. The Figure 3. describes the process.

Multiple layers of the subgraph networks are stacked, with different weights assigned to **genc**() in each layer. To obtain polyline-level features, we compute:

$$p = \phi_{agg}(v_i^{L_p}) \tag{4}$$

where $\phi_{agg}(\bullet)$ is maxpooling.

Our polyline subgraph network can be considered as an extension of PointNet. When we set the start point ds equal to the end point de and leave the attribute a and label l empty, our network shares the same inputs and computation flow as PointNet. However, our method goes beyond PointNet by incorporating ordering information into vectors, constraining the connectivity of subgraphs based on polyline groupings, and encoding attributes as node features. These enhancements make our approach well-suited for encoding structured map annotations and agent trajectories, allowing us to effectively capture the relationships and contextual information present in the data.

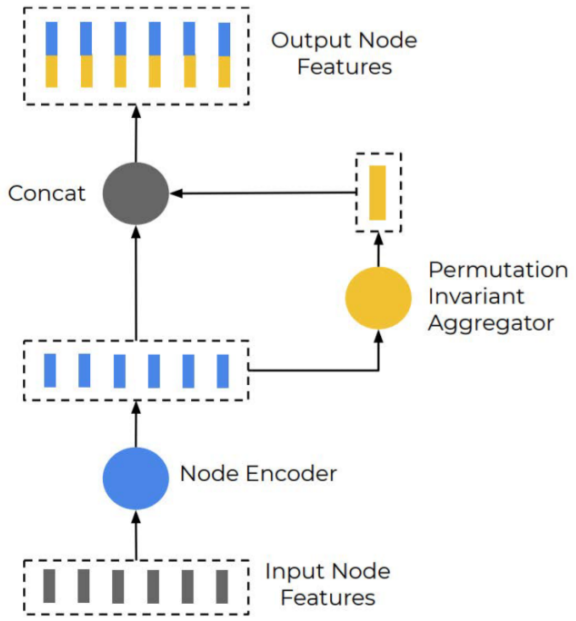

Fig. 3. This figure desThe computation flow on the vector nodes of the same polyline. [3]

E. Policy Model

Every high-level object, such as an agent, lane, or crosswalk, comprises a specific number of points, each with a feature dimension denoted as F. These individual points undergo embedding into a 128-dimensional space. To enhance the model's understanding of order, we introduce sinusoidal embeddings to each object's points. Subsequently, we input this data into our PointNet[15] implementation, which consists of three PointNet layers, ultimately yielding a descriptor of size 128 for each object. Following this, we apply a single layer of scaled dot-product attention: here, the feature descriptor corresponding to the ego object serves as the key, while all other feature descriptors function as keys and values. We introduce an additional type embedding into the keys, allowing the model to attend to values based on object types. Finally, we employ a multi-layer perceptron (MLP) to project the output into the desired shape, which is typically $T \times 3$ for a trajectory of length T, with each step described by xy position and yaw angle.

We define the state as a comprehensive set of both static and dynamic components that serve as inputs to the model. Each component comprises a variable number of points, which can encode temporal information (e.g., for agents) and spatial information (e.g., for lanes). The number of features per point varies according to the type of component. To ensure uniformity for the initial fully connected layer, we pad all features to a fixed size denoted as F. We encompass all these components within a circular field of view (FOV) with a radius of 35 meters around the SDV. It's worth noting that for the sake of efficiency and simplicity, we execute this query just once and subsequently build the world state based on this initial information.

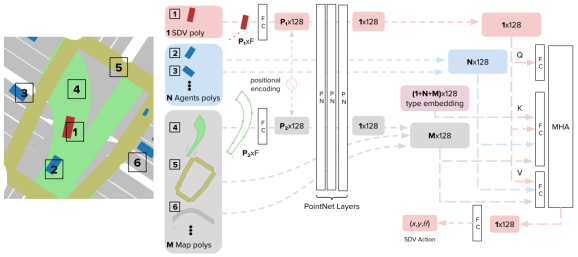

Fig. 4. Inspired by [14] and [3],here is overview of our policy model. Each element in the state is independently forwarded to a set of PointNet layers. The resulting features go through a Multi-Head Attention layer which takes into account their relations to output the final action for the SDV. The bird's-eye-view image on the left is only for illustrative purposes.

F. Policy Iteration

The input data is encoded in an ego-centric frame of reference where the SDV is always at a fixed location relative to a frame. The input to our model consists of:

1) *SDV*: the current and past poses and sizes of the SDV
2) *Agents*: the current and past poses of perceived agents, their sizes, and object type (e.g. vehicle, pedestrian, cyclist) produced by the SDV's perception system.
3) *Static map elements*: road network from High Definition (HD) maps including lanes, cross-walks, stop lines, localized using the SDV's localization system.
4) *Dynamic map elements*: traffic light states, and static obstacles detected by the perception system (e.g. construction zones).
5) *Route*: the intended global route that the SDV should follow.

Defining a trajectory τ as a sequence of T discrete states separated uniformaly in time by Δt. Each state s_t is defined as:

$$s_t = \{x_t, y_t, \theta_t\} \tag{5}$$

where x_t, y_t, θ_t correspond to the pose of the rear axle of the SDV with respect to a fixed coordinate frame at time t.

We define imitation learning problem as minimizing the L1 pose distance, denoted as $L(s_t, a_t) = \|\bar{p}_t - p_t\|$, between the expert and learner on a sequence of collected real-world demonstrations. $\bar{p}_t$ and p_t represent the expert and learner poses respectively. This can be expressed as a discounted cumulative expected loss on the set of collected expert scenarios:

$$J(\pi) = \mathbb{E}_{\bar{\tau}\sim\pi_E}\, \mathbb{E}_{\tau\sim\pi} \sum_t \gamma^t L(s_t, a_t) \tag{6}$$

By optimizing the loss function, we aim to align the trajectory of learned policy as closely as possible with that of the expert. Additionally, we ensure that the trajectory remains within the effective region of the simulator's approximation.

This gradient can be computed for deterministic policies π using backpropagation through time leveraging the differentiability of the simulator S. The equation (7) and (8) is the computation of policy gradient J_θ around the rollout trajectory.

$$J_s^t = L_s + L_a\pi_s + \gamma J_\theta^{t+1}(S_s + S_a\pi_s) \tag{7}$$

$$J_\theta^t = L_a\pi_\theta + \gamma(J_s^{t+1}S_a\pi_\theta + J_\theta^{t+1}) \tag{8}$$

IV. Results and Evaluation

A. Training

The training metrics are as follow:

1) *Average displacement error*
2) *Average heading error*
3) *Final displacement error*
4) *Final heading error*

We exclusively train the model using a minidataset from nuPlan. During training, our evaluation criterion is the Mean Absolute Error (MAE), serving as the loss metric to gauge how closely the model's predictions align with expert trajectories. We employ the Adam Optimizer with a learning rate of 0.0001. It's worth noting that the initial default value of the learning rate was relatively high, which resulted in the validation loss converging quite rapidly. The training took approximately 60 minutes per epoch when trained on a Nvidia RTX 3070 8GB GPU for a training set of around 5000 training steps.

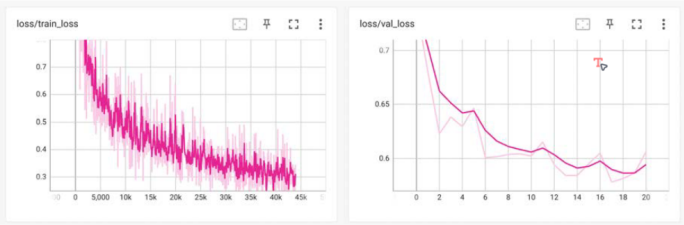

Fig. 5. This figures describes Train and Val loss within 20 Epochs. We could see ML model is converged after 16 epochs.

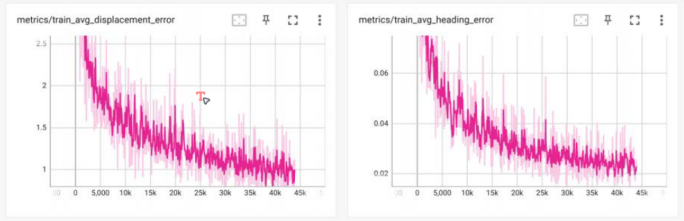

Fig. 6. This figures describes Average displacement error and average heading error in training steps.

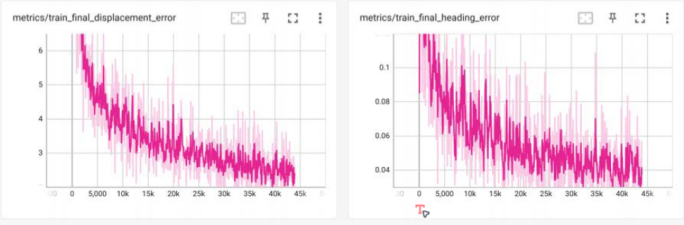

Fig. 7. This figures describes Average displacement error and average heading error in validation steps.

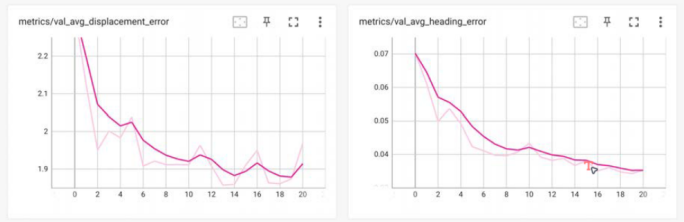

Fig. 8. This figures describes Final displacement error and final heading error in training steps.

B. Evaluation

We compare our proposed algorithm against two state-of-the-art baselines:

- SimplePlanner: The planner plans a straight line with constant speed.
- Intelligent Driver Model Planner(IDMPlanner): The planner consists of two parts, namely, path planning and longitudinal control(IDM policy).

The available metrics in nuPlan to evaluate scenarios and score planners are described in the table I and table II.

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	Histogram of: • miss rate (ratio)
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)

TABLE I
METRICS IN PART I AND DETAILED DESCRIPTION

C. Result

Following the introduction of baselines and metrics, we employ the nuPlan board to evaluate various planners. We assess three planners across 31 scenarios, encompassing situations such as high jerk, proximity to multiple vehicles, pickup and drop-off points, and unprotected crossings. The resulting scores will be presented in the upcoming three figures.

Metric	Thresholds	Visualization
No at fault collision	max violation threshold vru = 0 max violation threshold vehicle = 0 max violation threshold object = 1	Histogram of: • number of at fault collisions • min/max/mean of at fault collisions energy, if exists (m/s)
Drivable Area Compliance	max violation threshold = 0.3m	Histogram of: • boolean (True if there is no drivable area violation)
Driving Direction Compliance	driving direction violation threshold = 2m driving direction violation threshold = 6m time horizon = 1s	Histogram of: • score value
Making progress	min progress threshold = 0.2	Histogram of: • boolean (True if progress ratio is more than min progress threshold)
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)
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 = 0.1m	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
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

TABLE II
METRICS IN PART 2 AND 3 AND DETAILED DESCRIPTION

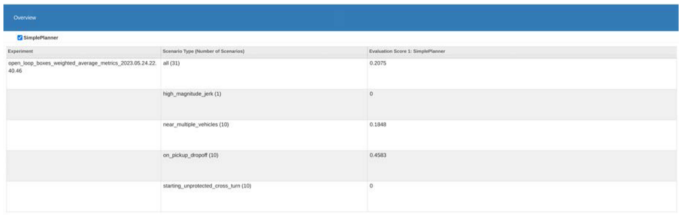

Fig. 9. This figure describes scores of simple planner in 31 scenarios.

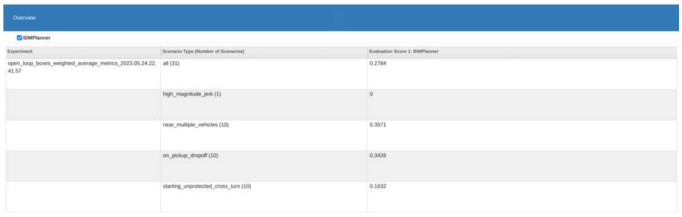

Fig. 10. This figure describes scores of IDMplanner in 31 scenarios.

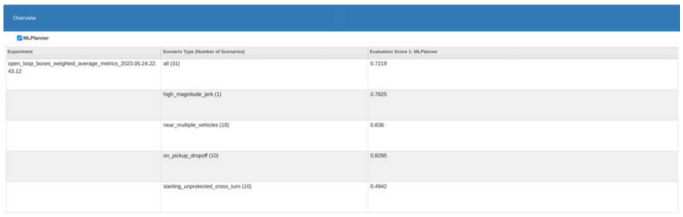

Fig. 11. This figure describes scores of ML planner in 31 scenarios.

Following the scoring test, the ML planner exhibited superior performance when compared to the IDM planner and the simple planner in a range of scenarios. The ML planner attained the highest score of 0.74, with the maximum score of 1.0 indicating optimal performance. In contrast, the IDM planner received a score of 0.27, while the simple planner yielded a score of 0.2.

In the scoring test, the ML planner demonstrated superior performance compared to the IDM planner and the basic planner. It particularly excelled in scenarios with high jerk, situations involving multiple vehicles, pickup and drop-off tasks, and starting in unprotected cross scenarios. This outstanding performance underscores its advanced capabilities in maneuvering, situational awareness, and decision-making. Ultimately, this leads to improved driving safety and efficiency.

V. NUPLAN DATASET

nuPlan datasets collect approximately 1200h of driving data from 4 cities, namely Boston, Pittsburgh, Las Vegas and Singapore. About 838h are from Las Vegas, with the remaining data split equally across the other cities. We drive on the Las Vegas strip between the Wynn and Aria casinos, as well as between Paris and Luxor. The Las Vegas dataset resembles that of a typical robotaxi application, where the route is known in advance[16].

nuPlan data aims for a diverse set of driving conditions, but exclude heavy rain and night data. All data is collected via manual driving to get suitable ground-truth for imitation learning. The vehicle operators (VOs) are instructed to drive safely, confidently and assertively. To make this data useful for planning, VOs drive goal oriented. This means that they define a goal (pick-up and drop-off points, bus stops, parking lots) and then drive a direct route towards the goal.

The following sensors are used:

- 5x lidar (20Hz)
  - 2x Pandar 20P (front/rear bumper)
  - 2x Pandar 40P (left/right A-pillar)
  - 1x Pandar 40PT (roof)
- 8x camera (10Hz)
  - D3 Engineering D3RCM
  - Sony IMX390, 2000 x 1200 resolution, split-pixel image sensor
- 1x Honeywell HG1120 IMU (100Hz)
- 1x Trimble BX992 (20Hz)

All sensors are calibrated with respect to their extrinsics and intrinsics (cameras only). Cameras and lidars are synchronized with each other.

- log: Information about the log from which the data was extracted.
- ego-pose: Ego vehicle pose at a particular timestamp.
- camera: The camera table contains information about the calibration and other settings of a particular camera in a particular log.
- image: The image table stores metadata to retrieve a single image taken from a camera. It does not store the image itself.
- lidar: The lidar table contains information about the calibration and other settings of a particular lidar in a particular log.
- 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.

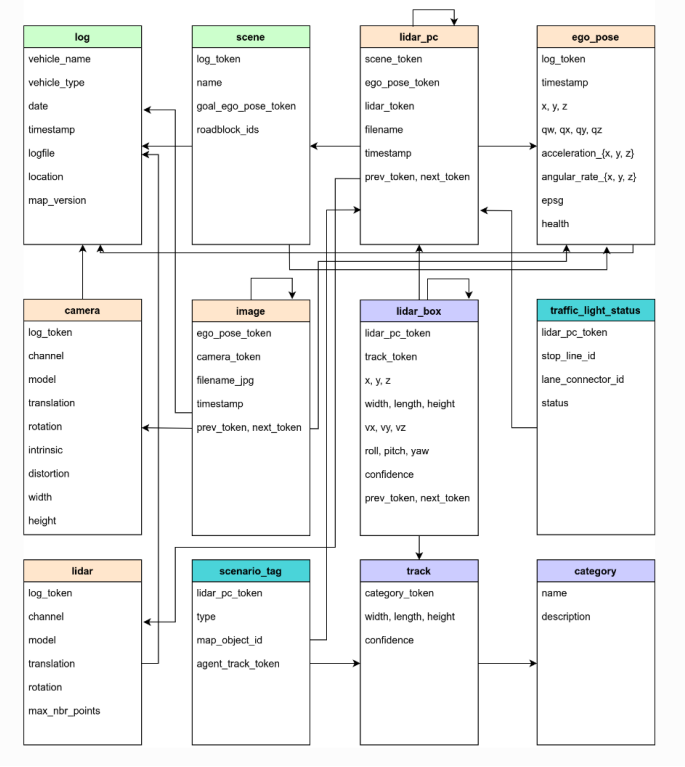

Fig. 12. This figures describes the database schema used in nuPlan. All annotations and metadata (including calibration, maps, vehicle coordinates etc.) are covered in relational databases.

- 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.
- track: An object track, e.g. particular vehicle. This table is an enumeration of all unique object instances that observed.
- category: Taxonomy of object categories (e.g. 'vehicle', 'bicycle', 'pedestrian', 'traffic-cone', 'barrier', 'czone-sign', 'generic-object').
- 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.
- 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).
- traffic-light-status: The observed motion of agents in the environment is used 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.

VI. Docker

Docker is an open platform for developing, shipping, and running applications. Docker enables developers to separate the applications from own infrastructure so developers can deliver software quickly. With Docker, developers can manage the infrastructure in the same ways manage the applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code, developers can significantly reduce the delay between writing code and running it in production[17].

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security lets developers to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so developers don't need to rely on what's installed on the host.

A. Docker's Usages

1) *Fast, consistent delivery of your applications*
   Docker streamlines the development lifecycle by allowing developers to work in standardized environments using local containers which provide your applications and services. Containers are great for continuous integration and continuous delivery (CI/CD) workflows.
2) *Responsive deployment and scaling*
   Docker's container-based platform allows for highly portable workloads. Docker containers can run on a developer's local laptop, on physical or virtual machines in a data center, on cloud providers, or in a mixture of environments.
   Docker's portability and lightweight nature also make it easy to dynamically manage workloads, scaling up or tearing down applications and services as business needs dictate, in near real time.
3) *Running more workloads on the same hardware*
   Docker is lightweight and fast. It provides a viable, cost-effective alternative to hypervisor-based virtual machines, so you can use more of your server capacity to achieve your business goals. Docker is perfect for high density environments and for small and medium deployments where you need to do more with fewer resources.

B. Docker Architecture

Docker uses a client-server architecture. The Docker client talks to the Docker daemon, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon can run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

1) *The Docker daemon*
   The Docker daemon ('dockerd') listens for Docker API requests and manages Docker objects such as images,

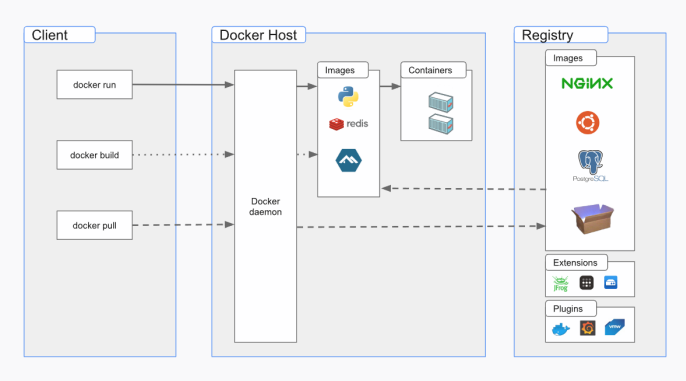

Fig. 13. This figures describes the Docker architectures.

containers, networks, and volumes. A daemon can also communicate with other daemons to manage Docker services.

2) *The Docker client*
The Docker client(”docker) is the primary way that many Docker users interact with Docker. When you use commands such as ’docker run’, the client sends these commands to ’dockerd’, which carries them out. The ’docker’ command uses the Docker API. The Docker client can communicate with more than one daemon.

3) *Docker registries*
A Docker registry stores Docker images. Docker Hub is a public registry that anyone can use, and Docker looks for images on Docker Hub by default. You can even run your own private registry.
When you use the ’docker pull’ or ’docker run’ commands, Docker pulls the required images from your configured registry. When you use the ’docker push’ command, Docker pushes your image to your configured registry.

4) *Images*
An image is a read-only template with instructions for creating a Docker container. Often, an image is based on another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.
You might create your own images or you might only use those created by others and published in a registry. To build your own image, you create a Dockerfile with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

5) *Containers*
A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.
By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container’s network, storage, or other underlying subsystems are from other containers or from the host machine.
A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that aren’t stored in persistent storage disappear.

VII. Conclusion

In summary, the GNN framework represents a huge forward in autonomous driving technologies. By harnessing the capabilities of imitation learning and real-world demonstrations, this approach empowers autonomous vehicles to acquire intricate driving skills and navigate urban settings with resilience and adaptability. The results of this study underscore the efficacy of our self-developed GNN model in learning from a wide array of real-world scenarios, surpassing the constraints of conventional manually-crafted rules and heuristics.

The integration of policy gradients and deep neural networks within our self-developed GNN framework facilitates the acquisition of driving behaviors closely mirroring human expertise. This not only augments the performance of autonomous vehicles but also bolsters the safety and efficiency of urban transportation as a whole. By integrating knowledge and experience from real-world demonstrations, our GNN model benefits from the accumulated wisdom of human drivers, enabling it to handle diverse challenging situations, including lane changes, interactions with traffic lights, and intricate intersections.

Furthermore, the scalability and versatility inherent in our self-developed GNN framework hold considerable promise for future advancements. As research progresses, the exploration of efficient representation learning techniques, the integration of advanced perception systems, and the incorporation of transfer learning and domain adaptation methods can further enhance the capabilities of autonomous vehicles in a variety of urban environments. The ongoing refinement of our self-developed GNN model can pave the way for heightened safety, diminished traffic congestion, and an overall improved driving experience.

References

[1] Bansal, Mayank, Krizhevsky, Alex, Ogale, Abhijit. (2018). ChauffeurNet: Learning to Drive by Imitating the Best and Synthesizing the Worst.
[2] Houston, J., Zuidhof, G., Bergamini, L., Ye, Y., Chen, L., Jain, A., Omari, S., Iglovikov, V., and Ondruska, P. “One thousand and one hours: Self-driving motion prediction dataset”. In: Conference on Robot Learning. PMLR. 2021, pp. 409– 418.
[3] Gao, J., Sun, C., Zhao, H., Shen, Y., Anguelov, D., Li, C., and Schmid, C. “Vectornet: Encoding hd maps and agent dynamics from vectorized representation”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recogni- tion. 2020, pp. 11525–11533.
[4] Liang, M., Yang, B., Hu, R., Chen, Y., Liao, R., Feng, S., and Urtasun, R. “Learning lane graph representations for motion forecasting”. In: Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16. Springer. 2020, pp. 541–556.

[5] Caesar, H., Kabzan, J., Tan, K. S., Fong, W. K., Wolff, E., Lang, A., Fletcher, L., Beijbom, O., and Omari, S. “nuplan: A closed-loop ml-based planning bench- mark for autonomous vehicles”. In: arXiv preprint arXiv:2106.11810 (2021).

[6] Buehler, M., Iagnemma, K., and Singh, S. The DARPA urban challenge: autonomous vehicles in city traffic. Vol. 56. springer, 2009.

[7] Wei, J., Snider, J. M., Gu, T., Dolan, J. M., and Litkouhi, B. “A behavioral planning framework for autonomous driving”. In: 2014 IEEE Intelligent Vehicles Symposium Proceedings. IEEE. 2014, pp. 458–464.

[8] Chang, Y., Omari, S., and Vitelli, M. S. Systems and methods for determining vehicle trajectories directly from data indicative of human-driving behavior. US Patent App. 16/706,307. June 2021.

[9] Zeng, W., Luo, W., Suo, S., Sadat, A., Yang, B., Casas, S., and Urtasun, R. “Endto-end interpretable neural motion planner”. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 8660–8669.

[10] Lyu, H., Sha, N., Qin, S., Yan, M., Xie, Y., and Wang, R. “Advances in neural information processing systems”. In: Advances in neural information processing systems 32 (2019).

[11] Ross, S., Gordon, G., and Bagnell, D. “A reduction of imitation learning and structured prediction to no-regret online learning”. In: Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLRWorkshop and Conference Proceedings. 2011, pp. 627–635.

[12] Shalev-Shwartz, S., Shammah, S., and Shashua, A. “Safe, multi-agent, reinforcement learning for autonomous driving”. In: arXiv preprint arXiv:1610.03295 (2016).

[13] Ziebart, B. D., Maas, A. L., Bagnell, J. A., Dey, A. K., et al. “Maximum entropy inverse reinforcement learning.” In: Aaai. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[14] Scheel, O., Bergamini, L., Wolczyk, M., Osi´nski, B., and Ondruska, P. “Urban driver: Learning to drive from real-world demonstrations using policy gradi- ents”. In: Conference on Robot Learning. PMLR. 2022, pp. 718–728.

[15] Qi, Charles R., Su, Hao, Mo, Kaichun and Guibas, Leonidas J. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. (2016). , cite arxiv:1612.00593Comment: CVPR 2017 .

[16] https://www.nuscenes.org/nuplan

[17] https://docs.docker.com/