2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

HP^3: Hierarchical Prediction-Pretrained Planning for Unprotected Left Turn

Zhihao Ou[†,1], Zhibo Wang[†,1], Yue Hua[1], Jinsheng Dou[2], Di Feng[2] and Jian Pu[*,1]

***Abstract*— Trajectory planning for unprotected left turns poses a significant challenge in autonomous driving. Reinforcement learning (RL) offers potential, but existing methods often rely on scenario-specific state representations, limiting their adaptability. This paper introduces Hierarchical Prediction-Pretrained Planning (HP^3), a generalizable hierarchical RL framework designed for unprotected left turns. HP^3 leverages historical trajectories of all vehicles and complete map information to achieve versatile state representation and generalizable scene understanding. Its two-layer architecture predicts semantic behavior (upper layer) and generates corresponding trajectories (lower layer). A scene encoder comprehends trajectories and roads, while a trajectory decoder outputs sequential points. To accelerate convergence, we pretrain the main network on a modified trajectory prediction dataset. Evaluation on a CARLA-based map with complex, unprotected left-turn intersections demonstrates HP^3's superiority over rule-based and simple RL-based methods, highlighting the effectiveness of our pretraining approach for this critical autonomous driving task.**

I. INTRODUCTION

Trajectory planning is a cornerstone of autonomous driving, demanding sophisticated solutions to address its inherent complexities. Reinforcement learning (RL) presents a compelling avenue for this task, enabling autonomous vehicles to develop adaptive decision-making skills through iterative environmental interaction. However, current RL approaches often rely on problem-specific state representations, hindering their generalization to diverse scenarios. The challenge lies in introducing a more versatile state representation and generalizable scene understanding to enhance the adaptability of planning frameworks.

To achieve the above versatile state representation and generalizable scene understanding, a powerful network architecture is essential. Recent advances in trajectory prediction have demonstrated the superior representation ability of vetorized data and corresponding network architecture based on GNN or Transformer[1], [2], through which the value of incorporating vehicle trajectories and HD maps for accurate scene analysis is proved. However, due to the sparsity of the reward constructing, the value function in reinforcement learning is usually unsmooth [3]. But deep neural networks may have difficulty in fitting irregular functions [4]. Therefore, integrating such complex networks into RL risks hindering convergence.

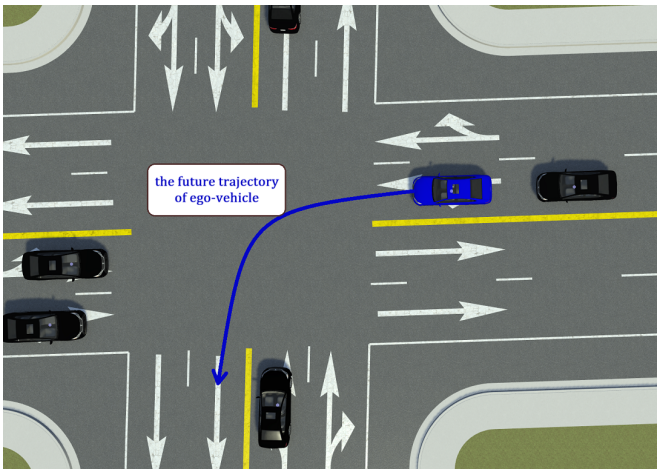


Fig. 1: An illustration of unprotected left turn at an intersection without traffic signals. The ego vehicle in blue must navigate the maneuver with unknown intents of the black vehicles. The blue line shows the ego vehicle's planned trajectory to accomplish this challenging scenario, requiring generalizable scene understanding and planning.

In this paper, we bridge these gaps by introducing Hierarchical Prediction-Pretrained Planning (HP³), a novel hierarchical reinforcement learning framework tailored for unprotected left turns (Figure 1). HP³ leverages a versatile state representation derived from historical trajectories of all perceptible vehicles and detailed map information. Its two-layer architecture includes a behavior planner that generates semantic behaviors and a trajectory planner that translates them into a reference trajectory. We employ a Transformer-based scene encoder for generalizable scene comprehension. Moreover, inspired by studies demonstrating the power of pretraining[5], we incorporate a modified trajectory prediction pretraining method to accelerate convergence. This strategic pretraining leverages the strong link between trajectory prediction and planning, potentially enhancing both scene understanding and convergence speed.

In summary, the main contributions of this paper are:

- Proposing a hierarchical reinforcement learning framework for unprotected left turns, utilizing generalizable scene understanding and versatile state representation.
- Introducing a shared scene encoder to deal with sequential input of indefinite length as well as a trajectory decoder to generate sequential trajectory points.
- Delivering a prediction-task-based pretraining method

[†]Equal contribution.
[*]Corresponding author.
[1]Fudan University, Shanghai, China, {zh_ou21, zbwang21, yhua22}@m.fudan.edu.cn, jianpu@fudan.edu.cn
[2]Mogo Auto Intelligence and Telematics Information Technology Co., Ltd, Beijing, China, doujinsheng@dingtalk.com, fd940056938@163.com

to the network to improve convergence outcomes.
- Demonstrating through extensive experiments that our planning model achieves superior performance compared to both traditional rule-based approach and simple HRL method.

II. RELATED WORKS

A. Hierarchical Reinforcement Learning in Planning for Autonomous Driving

Planning in autonomous driving can be decomposed into several subtasks, including behavior planning, trajectory planning, and motion planning. Recently, various AI-based methods have been applied to planning tasks [6]. [7], [8], [9], [10] have applied reinforcement learning (RL) to the task of behavior planning and shown great potential. [8] designed a modified encoder for unordered vehicles, learning from point cloud problems. It generalized the action space to both lateral and longitudinal behaviors, including lane changing and speed controlling, and outperformed rule-based methods.

Different from traditional non-hierarchical reinforcement learning based on the Markov decision process (MDP), hierarchical reinforcement learning (HRL) is based on semi-MDP (SMDP), a generalization of MDP [11]. [12] provided a method to generate options over actions. An HRL method was proposed to learn task-agnostic options through self-supervised learning. They demonstrated the superiority of their method compared to regular RL methods on sparse-reward robotic manipulation and navigation tasks.

HRL methods have numerous applications for autonomous vehicle planning as well, such as stopping at an intersection [13], [14] and lane changing [15], [16]. [14] proposed an HRL-based behavioral planning architecture with hybrid reward mechanisms for different hierarchies, which can perform an autonomous vehicle planning task in a simulated environment with several subgoals. [15] showed us the possibility of combining rule-based methods with RL methods. An HRL model was trained to assist the rule-based model so that their HRL planner outperformed both rule-based methods and pure data-driven methods in a stochastic and uncertain simulation environment.

Unprotected left turn is a more complicated autonomous driving scenario [17], [18], [19], where the agent needs to interact with vehicles coming from multiple directions. [17] applied HRL method to urban intersection scenarios, which are more complex than highway lane-changing scenarios. With behavior level segmentation, their model performed more effectively and quickly in left-turn scenarios in games with opposite vehicles.

B. Network Architecture Designing for Prediction and Planning

Trajectory prediction generates motion expectations for neighboring vehicles in the upcoming seconds. Early works such as Social LSTM [20] emphasized the sequential nature of trajectories and thus adopted Recurrent Neural Networks (RNNs) as their primary architecture. Additionally, Convolutional Neural Networks (CNNs) were also adopted due to CNN's better representation ability for geometric data [21].

However, both RNNs and CNNs struggle to adequately represent vectorization geometric information such as direction and speed. Thus, Graph Neural Networks (GNNs) based models have been proposed. Treating trajectories and roads as vertex and edges of a directed graph, GNN-based methods [22], [1] have reached outstanding performance on public datasets. On top of that, transformer-based methods were also proposed due to their strong information fusion capabilities. Previous works like [23], [2], [24] have demonstrated that transformer-based models are able to secure superior accuracy as well as high efficiency.

Several works have made primary attempts to adopt prediction-like network to planning tasks. [25] introduced Deep Recurrent Deterministic Policy Gradient to deal with longer input sequences and to explore the time-dependent. Recursive methods were introduced for better estimation of the underlying environmental state, while LSTM was used as the basic RNN structure. [26] used the history of state observations and LSTM layers in their model to compensate for observation noise in state space. Experiments showed that their framework improved upon existing methods in several aspects.

III. METHODS

We propose HP^3, an RL-based hierarchical planner for urban autonomous driving, as shown in Figure 2. HP^3 consists of a high-level behavior planner and a low-level trajectory planner. The behavior generated by the upper layer activates the corresponding trajectory generator in the lower layer and therefore output trajectories with specific behavior. Deep Q-Network (DQN) and Deep Deterministic Policy Gradient (DDPG) are adopted to the behavior planner and the trajectory planner respectively.

A. Preliminaries

1) Scene representation: In HP^3, the observation space comprises the ego vehicle's trajectory, neighbor vehicles' trajectory and HD map. For vehicles, we log their historical trajectory rather than their current motion states. Furthermore, all the other nearby vehicles are taken into consideration equally. As for map representation, the planner takes the entire HD map into account, instead of just a few local position parameters.

Different from raster form, vector data has advantages in representing geometric features like direction and velocity [27]. Therefore, vehicle trajectory and HD map are all unified as vectors [24]:

$$v^k = [x_k, y_k, x_k^{pre}, y_k^{pre}, \Delta x, \Delta y, v_{id}, s_{id}] \ , \qquad (1)$$

where $[x_k, y_k]$ and $[x_k^{pre}, y_k^{pre}]$ are respectively the end and start position of the vector v^k, $[\Delta x, \Delta y]$ is the position change, and the detail setting of $[v_{id}, s_{id}]$ will be clarified in III-B.1.

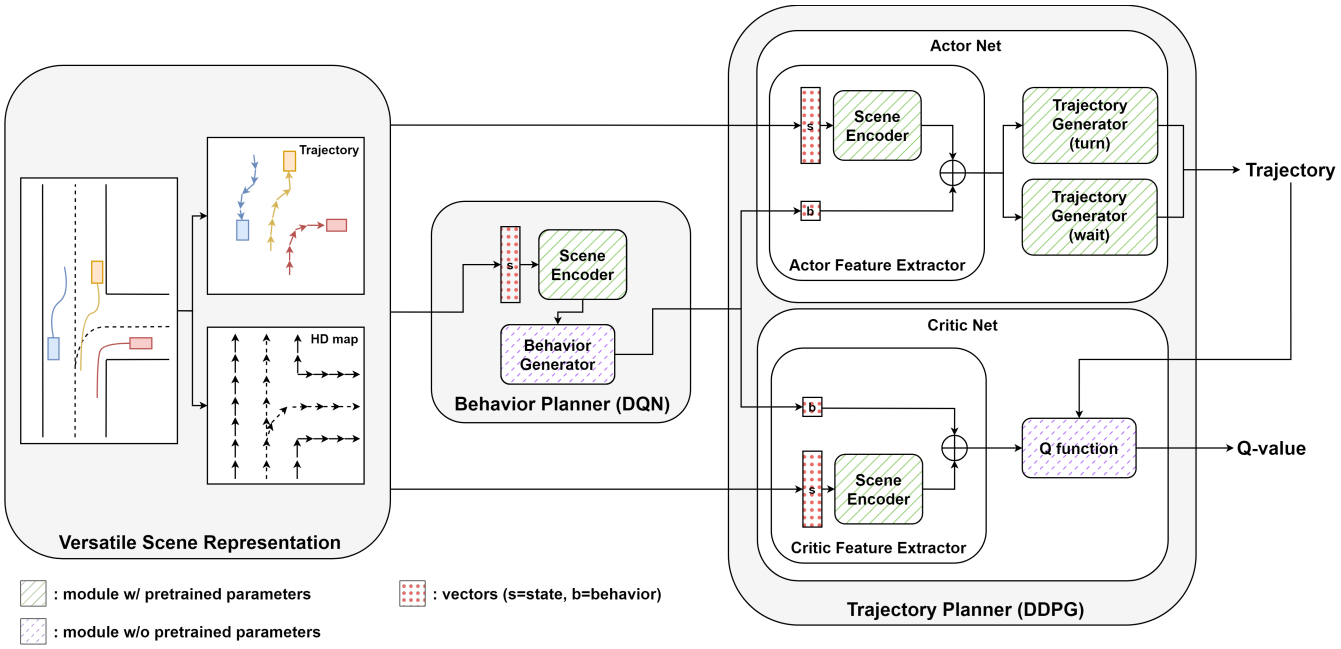


Fig. 2: The overall framework of hierarchical prediction-pretrained planning (HP^3). DQN and DDPG are respectively adopted to the behavior planner and trajectory planner.

It's worth noting that the representation of both observations and future waypoints are transformed to the local coordinate system of the ego vehicle.

2) Reward function: The reward function consists of indicators of safety, efficiency, and comfort. For the unprotected left turn scenario, the reward function is designed as the summation of following components:

$$\begin{aligned} r = &- \lambda_1 \times \mathbb{I}(collision) - \lambda_2 \times |\alpha| v^2 \\ &+ \lambda_3 \times \left\{ v / v_{max} - 1 \right\} - \lambda_4 \times \mathbb{I}(v > v_{max}) \\ &+ \lambda_5 \times sgn(b_{plan} = b_{traj}) \ . \end{aligned} \tag{2}$$

The first term is a collision penalty designed for driving safety. The second term is for passenger comfort, where $|\alpha|v^2$ is the lateral acceleration and α is the steering angle in radians. The third and fourth terms encourage the passing efficiency by urging the agent to drive as fast as possible adhering to speed limits. Additionally, we impose a behavior reward, the last term in Equation 2, to ensure alignment between the vehicle's driving trajectory and its intended driving behavior. Here, b_{plan} represents the behavior predicted by the behavior planner, while b_{traj} represents the vehicle's actual observed driving behavior. Specifically, we set λ_1 = 200, λ_2 = 0.01, λ_3 = 1, λ_4 = 10, λ_5 = 150.

B. Behavior Planner

1) Reinforcement learning model: DQN is a generalization of Q-learning to continuous state spaces. For Q-learning [28], the action-value function $Q(s, a)$ refers to the estimated sum of discounted rewards while performing action a at state s with discount factor γ and policy π. Instead of using a Q-table, a Deep Neural Network (DNN)-inspired Q-net is proposed to fit the Q-values of different actions over a continuous state space. The state representation of the behavior planner $s^{behavior}$ consists of both vehicle trajectories and the HD map:

$$\begin{aligned} s^{behavior} = (&s_0^{traj}, s_1^{traj}, \ldots, s_m^{traj}; \\ &s_1^{lane}, \ldots, s_n^{lane}) \ , \end{aligned} \tag{3}$$

where s_0^{traj} and s_i^{traj} stands for the historical trajectory of ego vehicle and target vehicle i respectively, and s_j^{lane} stands for the polyline of lane j, while there exist m other vehicles and n lanes in total within the perception range. In detail, both s^{traj} and s^{lane} are lists of vectors defined in Equation 1. Specifically, v_{id} stands for the chronological index and s_{id} stands for the vehicle index in s^{traj}, while v_{id} stands for the directional index and s_{id} stands for the lane index in s^{lane}:

$$\begin{aligned} s_i^{traj} &= [v_{traj_i}^k], k = 0, 1, \ldots \ , \\ s_j^{lane} &= [v_{lane_j}^k], k = 0, 1, \ldots \ . \end{aligned} \tag{4}$$

The action space for the behavior planner consists of two different behaviors called *turn* and *wait*, based on whether the ego vehicle needs to let another vehicle pass in advance or not. For ease of understanding, the action of the behavior planner will be represented by *behavior* or b, hereafter:

$$b \in \{Turn, Wait\} \ . \tag{5}$$

2) Network architecture: The Q-net of the behavior planner consists of a scene encoder inspired by the previous trajectory prediction research [22], [1], [24] and a behavior generator, as shown in Figure 2. The vectorized data is encoded by the scene encoder shown in Figure 3. In detail,

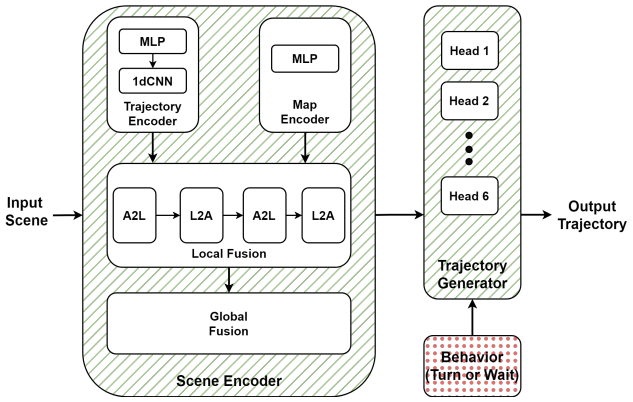


Fig. 3: Structure of the pretrained modules in HP[3], consisting of a scene encoder and multi-head trajectory generators.

trajectory and map data are encoded separately by the corresponding MLP encoder, with a 1d CNN adopted in the trajectory encoder for smoothing and denoising. Then, both trajectory and map encoded data are fused by a local fusion module, consisting of 4 transformer decoder blocks. A2L blocks calculate the cross attention from vehicles to lanes, while L2A blocks do the opposite. All the agents and lanes features are finally fused together by a global fusion module, which is a self-attention block.

Connecting with the scene encoder, an MLP-based behavior generator outputs the suggested behavior of the ego vehicle for the next time step. Specifically in the unprotected left turn scenario, behaviors are divided into 2 categories, *wait* or *turn*, based on whether the ego vehicle needs to let another vehicle go through first or not. The parameters of the scene encoder θ^{se} are pretrained through the method detailed in III-C.3, while the behavior generator is not be pretrained. In order to improve inference efficiency, the behavior planner and the actor net of the trajectory planner share the weights of the scene encoder θ^{SE} in HRL-based fine-tuning.

C. Trajectory Planner

1) Reinforcement learning model: In order to generalize DQN to continuous action spaces, DDPG builds a magic function to generate an action, playing the role of the actor model in Actor-Critic methods. While the Q-net of DQN takes only state s as input and outputs estimated $Q(s, a_i)$ corresponding to each possible action a_i, the Q-net of DDPG, playing the role of critic model in Actor-Critic methods, takes both state s and action a as input, and outputs an estimated q of Q-value $Q(s, a)$. In this work, the state representation of the trajectory planner s^{traj} consists of the same scene representation as $s^{behavior}$ and the behavior b offered by the behavior planner:

$$s^{traj} = (s^{behavior}, b) \ , \tag{6}$$

while the action for the trajectory planner is the planned future waypoints of the ego:

$$a^{traj} = [[x_k, y_k]], k = 1, 2, \dots \ . \tag{7}$$

Hence, the actor and critic model are described as:

$$a^{traj} = A_{\theta_A}(s^{behavior}, b), q = Q_{\theta_Q}(s^{behavior}, b; a^{traj}) \ . \tag{8}$$

2) Network architecture: Both actor and critic nets [29] have their own feature extractor to transform the state into features. In the actor net, scene features are encoded by a same scene encoder as in behavior planner. The architecture of the trajectory generator is inspired by the decoder part in [22], [1], [24]. Specifically, as shown in Figure 3, multi-modal trajectories are generated based on the encoded features. Different heads are utilized to generate results with different semantic meanings (e.g. *turn* and *wait* in our setting). Future trajectory is generated by sending the scene features encoded by the scene encoder to the corresponding trajectory generator head to the behavior offered by the behavior planner.

In the critic net, the scene features are encoded by a separate scene encoder with the same architecture as that of the behavior planner and the actor net, while it doesn't share the weights with them in fine-tuning. The planned trajectory is encoded by an MLP-based encoder, and then concatenated to the scene features together with the behavior. An MLP-based Q-function net takes the concatenated features as input, and finally outputs the estimated Q value according to Equation 8. The parameters of the scene encoders θ^{SE} both in actor and critic net together with the parameters of the multi-head trajectory generator θ_i^{TG} are also pretrained according to III-C.3. However, the weights of the scene encoders don't remain the same in fine-tuning. The trajectory encoder and the Q-function net are not pretrained.

3) Pretraining on modified prediction task: Considering that both the input and output form of the actor net are approximately same as trajectory prediction models, we propose a pretraining method based on a modified trajectory prediction method. The pretraining method utilizes Argoverse2 Dataset [30], a widely-used trajectory prediction dataset with about 250k data cases, and about 20k cases crossing an intersection allocated from CARLA simulator as supplementary. For dataset modification, all the data cases are classified by a behavior label, which is *turn* or *wait* for left-turn cases according to our behavior setting, and *others* for go-straight and right-turn ones.

As shown in Algorithm 1, parameters of both scene encoder and trajectory generators are first trained by the modified Argoverse2 Dataset[30], and then, parameters of trajectory generators are further trained by the allocated CARLA dataset with the scene encoder frozen by learning rate α. The training process is derived from that of trajectory prediction, with a modified back-propagate policy. In a particular case, only one trajectory generator head is back propagated selected by the following rules:

$$i = \begin{cases} 1, & \textit{behavior} \text{ is } \textit{turn} \\ 2, & \textit{behavior} \text{ is } \textit{wait} \end{cases} \ , \tag{9}$$

for left-turn cases and

Algorithm 1 Pretraining on Modified Prediction Task

Initialize *Scene Encoder's* weights θ^{SE} from scratch.
Initialize *Trajectory Generators'* weights θ_i^{TG} from scratch.
for $epoch = 1 : max\_epoch$ **do**
    // Pretraining on Modified Argoverse2 Dataset
    for $case$ **in** Modified Argoverse2 Dataset **do**
        $\theta^{SE} \leftarrow \theta^{SE} - \alpha \cdot \nabla_{\theta^{SE}} L(\theta^{SE}, \theta_i^{TG})$
        $\theta_i^{TG} \leftarrow \theta_i^{TG} - \alpha \cdot \nabla_{\theta_i^{TG}} L(\theta^{SE}, \theta_i^{TG})$
    end for
end for
Freeze Scene Encoder's weights θ^{SE}.
for $epoch = 1 : max\_epoch$ **do**
    // Pretraining on Allocated CARLA Dataset
    for $case$ **in** Allocated CARLA Dataset **do**
        $\theta_i^{TG} \leftarrow \theta_i^{TG} - \alpha \cdot \nabla_{\theta_i^{TG}} L(\theta^{SE}, \theta_i^{TG})$
    end for
end for
Load pretrained θ^{SE} to all Scene Encoders.
Load pretrained θ_1^{TG}, θ_2^{TG} respectively to the corresponding Trajectory Generator.

$$i = argmin_k \ L(\theta^{SE}, \theta_k^{TG}), k = 3 : 6 \ , \tag{10}$$

for other cases.

The loss $L(\theta^{SE}, \theta_i^{TG})$ is determined by the trajectory that head i generates, consisting of a regression loss and a classification loss [24]:

$$L(\theta^{SE}, \theta_i^{TG}) = L_{reg} + L_{cls} \ . \tag{11}$$

All the pretrained parameters are fine-tuned to enhance the performance of the focusing scenario within the reinforcement learning stage.

IV. EXPERIMENTS

A. Experiment Settings

1) Simulation and training settings: To simulate real-world driving scenarios, we adopt CARLA simulator [31] in our experiment to train the reinforcement learning model. CARLA simulator is an open-source driving simulator based on Unreal Engine 4 (UE4). Thanks to the well-developed protocols, we can conveniently build our own training scenarios and collect necessary training data from the simulator.

In the experiments, we focus on the unprotected left turn scenario, one of the most challenging on-road scenarios [32]. Thus, instead of using the default map in CARLA, we develop a customized map featuring that there are 8 intersections with no traffic lights. The intersections differ in the crossing angles from each other, which brings a challenge to scene generalization.

Besides, to better evaluate our model, we define three different environment settings with varying difficulty levels: easy, medium and hard. Settings for these three environments are shown in Table I. The harder the environment, the more neighbor vehicles and the smaller range of spawn distance (distance to the intersection when turning starts) it has. In other words, It would be a denser traffic condition.

TABLE I: Environment settings for easy, medium, and hard conditions

Environment Type	Neighbor Vehicles	Spawn Distance
easy	3	(0, 20]
medium	3	(0, 10]
hard	5	(0, 5]

We use same settings for all of the following experiments. In detail, we train our model under three environment settings respectively for 2×10^5 time steps and test them on corresponding test sets, containing 1×10^4 steps of data. All of these data are generated by the CARLA simulator. Besides, all training and testing procedures are conducted with a frequency of 10 Hz, which means a new predicted trajectory would be generated every 0.1 second and a corresponding control signal would be applied to the ego vehicle.

We complete all of our experiments on a machine with an Intel(R) Core(TM) i7-11700K CPU, 64GB RAM, and an RTX 3080Ti GPU. The learning rate is set to 1×10^{-3} and the whole training procedure takes about 13 hours to coverage.

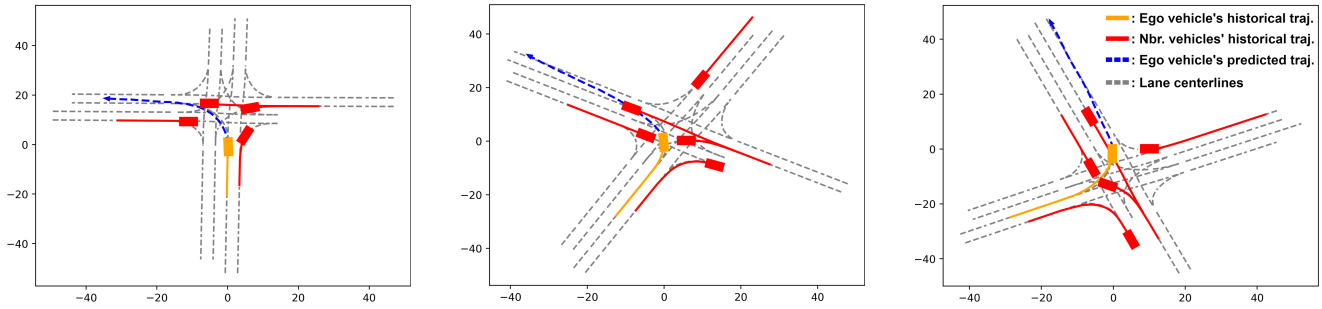
2) State and action spaces:

- State space: In the experiments, we apply a perception range of 80 meters. For trajectory data, we log the historical trajectories in the past 5 seconds for every vehicle in view and transform them into vector form. As for HD map, the road centerlines are divided into vectors with a granularity of 1 meter. Every 9 or less continuous lane vectors are regarded as a lane segment. Only lane vectors in view can be observed as well. All the vectors are expressed as the form in Equation 1.
- Action space: The trajectory planner will output the reference waypoints for the next 6 seconds at 10 Hz. Thus the action space is $\mathbb{R}^{60 \times 2}$. We set the value range in ± 300 meters in any dimension, which is ensured to be larger than the distance that a vehicle could reach in urban scenarios.

B. Overall Performance

We compare our model with the built-in autopilot controller in CARLA and a simple HRL method. The CARLA planner, as a typical rule-based method, builds a graph based on waypoints and searches for a route between the ego vehicle's current position and its destination. Once the route is determined, a control signal is generated and then applied to the ego vehicle. The simple HRL method replaces all the well-designed network architecture in HP^3 with MLPs.

As for metrics, we define three episode results named success, failure, and crash respectively. A successful episode means the ego vehicle has reached the target lane successfully and in time. A failed episode means the ego vehicle has gone into a wrong lane which corresponds to go-straight or right-turn, or run out of the time. A crashed episode means the ego vehicle has crashed with other vehicles or gone out



(a) Start (b) Turning (c) End

Fig. 4: Qualitative result from one case in a hard environment. Gray dashed lines represent the centerline of each lane in HD map. Orange, red, and blue respectively stand for ego vehicle's historical trajectory, neighbor vehicles' historical trajectory, and ego vehicle's planned future trajectory. Noting that the controller would only perform the first few waypoints with continuous re-planning. As shown in the figure, a left-turn trajectory is well proposed at the start, while the target lane is temporarily adjusted to avoid potential collisions. After the avoidance, the ego vehicle finishes its left turn following the planned trajectory to the target lane.

TABLE II: Quantitative results comparing with baselines

Method	Environment	Success Rate	Failure Rate	Crash Rate
Rule-based	Easy	83.18%	11.21%	5.61%
	Medium	66.20%	21.75%	12.05%
	Hard	25.70%	57.24%	17.06%
HRL(MLP)	Easy	84.20%	5.26%	10.54%
	Medium	41.68%	35.15%	23.17%
	Hard	35.14%	37.53%	27.33%
HP^3	Easy	90.15%	4.52%	5.33%
	Medium	81.63%	9.01%	9.36%
	Hard	57.91%	26.01%	16.08%

of the road. By counting the ratio of these three situations, we obtain the model's performance on unprotected left turn scenario.

1) Quantitative results: The quantitative results of HP^3 and baseline methods are shown in Table II. As shown in the table, HP^3 significantly outperforms both the CARLA built-in method and the simple MLP-based HRL method. Besides, the more challenging the environment, the greater the degree to which HP^3 leads the baselines in success rate. In the easy setting, our model is respectively 8.38% and 7.07% ahead rule-based and simple HRL method, while the margin grows to respectively 125.33% and 64.80% in the hard setting. The results demonstrate that HP^3's superiority in dealing with complex traffic conditions.

On the other hand, the rule-based model would bring many more failure cases compared with ours. Through visualization, we have found two major reasons for this phenomenon. First, vehicles guided by the rule-based method in the CARLA simulator sometimes obstruct each other and therefore cause a deadlock to the whole turning scenario and result in failure due to timeout. Besides, since the auto-pilot planner searches for the shortest path to the destination, it may cause ego vehicle to directly cut through other lanes and thus cause case failure because of traffic law violations. Both two phenomena are rare in results generated by HP^3, further proving HP^3's adaptability to complex turning scenarios.

2) Qualitative results: A visualization case from a hard environment is selected to illustrate the efficacy of our model. As shown in Figure 4, as the ego vehicle is at the edge of the intersection, the trajectory planner provides a normal left-turn trajectory to the closer target lane. Then, two neighbor vehicles approach the center of the intersection, causing a potential collision if the ego vehicle sticks to its previous planned trajectory. Under this circumstance, the trajectory planner provides a more aggressive trajectory, not only keeping away from the neighbor vehicle, but enabling the ego vehicle to pass in advance of both neighbor vehicles as well. It is in this way that the ego vehicle avoids potential collision and meanwhile guarantees the traffic efficiency. After passing by, the reference trajectory guides the ego vehicle into a more proper lane, following which the ego vehicle makes a successful left turn eventually.

C. Ablation Study

1) Effectiveness of the pretrained networks: We study the effects of the HP^3 network architectures by replacing the original encoder and generator structures with simpler multilayer perceptions. As shown in Table III, the networks designed for versatile state representation outperform the MLP alternatives in both scene encoder and trajectory generator. This indicates that the proposed architectures of HP^3 can bring a strong capability of generalizable scene understanding and provide useful spatio-temporal features to augment the planner.

Furthermore, in order to assess the influence of pre-training, we test the model both with and without loading pretrained weights, as shown in Table IV and Figure 5. The results demonstrate that adopting the pretraining on modified trajectory prediction task substantially improves the overall performance and coverage speed, which highlights the value of leveraging prior knowledge through pretraining on upsteam tasks.

The results are reasonable since the network architectures have been validated in prior trajectory prediction works. Moreover, displacing the scene encoder degrades more performance than displacing the trajectory generator. It reveals that the generalizable scene understanding capability is more critical than trajectory generation for planning tasks.

TABLE III: Ablation experiments for model structure

Scene Encoder		Trajectory Generator		Success Rate
Structure	Pretrained	Structure	Pretrained	
HP^3		HP^3		47.15%
MLP		HP^3	✓	48.02%
HP^3		HP^3	✓	62.74%
HP^3	✓	MLP		51.14%
HP^3	✓	HP^3		67.35%
HP^3	✓	HP^3	✓	81.63%

Results are from the environment of medium setting;
HP^3 refers to the corresponding model structure in Figure 3

TABLE IV: Ablation experiments for pretraining

Method	Success Rate	Failure Rate	Crash Rate
w/o pretraining	47.15%	14.38%	38.47%
w/ pretraining	81.63%	9.01%	9.36%

Results are from the environment of medium setting.

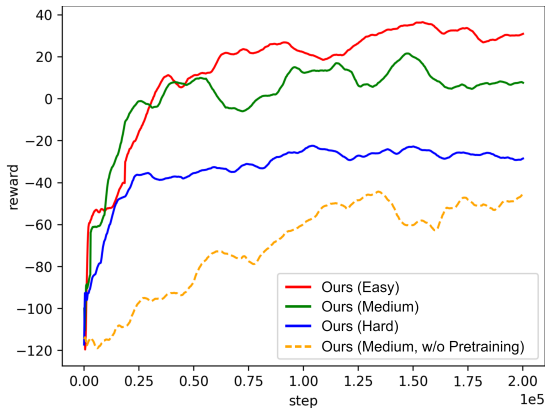


Fig. 5: The reward curve in training process. Red, green and blue solid lines stand for result from easy, medium and hard environments respectively. Orange dashed line stands for the result in medium environment without deploying pretraining methods.

TABLE V: Ablation experiments for hierarchical reinforcement learning structure

Method		Success Rate	Failure Rate	Crash Rate
DQN	DDPG			
		66.81%	14.85%	18.34%
	✓	77.98%	11.64%	10.38%
✓	✓	81.63%	9.01%	9.36%

Results are from the environment of medium setting.

2) Effectiveness of hierarchical reinforcement learning structure: To prove the efficacy of the hierarchical reinforcement learning structure of HP^3, we train a single DDPG-based trajectory planner without a DQN-based behavior planner and evaluate its performance. Besides, we evaluate the performance of the pretrained scene encoder and trajectory generators without fine-tuning as well.

As shown in Table V, the single trajectory planner performs worse than the full hierarchical framework, proving the effectiveness of behavior constructing. In addition, the non-fine-tuned model performs the worst, demonstrating that supervised learning may benefit from fine-tuning by reinforcement learning in specific tasks.

TABLE VI: Real-time applicability of HP^3.

Scene Encoder	Behavior Generator	Trajectory Generator	Total
13.292 ms	0.058 ms	0.796 ms	14.146 ms

Results are from the environment of hard setting.

D. Real-time Applicability

In terms of real-time applicability, we test our model for over 2000 timesteps. As shown in Table VI, HP^3 takes 14.146 ms in total to generate a reference trajectory from input on average. As announced at the end of III-B.2, the behavior planner and the trajectory planner share the weights of scene encoder, so that the planning module economizes about 13.292 ms accordingly, which could be utilized by high-consumption perception modules. Apart from the scene encoder, the behavior generator and the trajectory generator consumes 0.058 ms and 0.796 ms respectively.

V. CONCLUSIONS

This paper introduced Hierarchical Prediction-Pretrained Planning (HP^3), a novel hierarchical RL framework demonstrating significant performance gains in challenging scenarios. Its integration of a DQN-based behavior planner, DDPG-based trajectory planner, and a generalizable scene understanding module offers a unique solution. The modified trajectory prediction pretraining method further accelerates convergence. Experiments showcase HP^3's superiority over rule-based and simple RL methods, with ablation studies confirming the value of our architectural choices and pretraining.

Future work could focus on validating the superiority of HP^3 over optimization-based methods, and further exploring the integration of human-feedback reward shaping techniques. Moreover, the scenario generalization is a promising area for investigation. For example, the model could be tested on other intersections outside the training map, and efforts could be made to incorporate more trajectory generator heads for going straight and right turn. The model's success with unprotected left turns suggests its potential for handling even more complex scenarios, such as merging and pedestrian-vehicle interactions. Ultimately, this work highlights the promise of combining prediction and planning within a hierarchical RL framework for safer and more adaptive autonomous driving systems.

REFERENCES

[1] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, "Vectornet: Encoding hd maps and agent dynamics from vectorized representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 11 525–11 533.

[2] Z. Huang, X. Mo, and C. Lv, "Multi-modal motion prediction with transformer-based neural network for autonomous driving," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 2605–2611.

[3] M. Vecerik, T. Hester, J. Scholz, F. Wang, O. Pietquin, B. Piot, N. Heess, T. Rothörl, T. Lampe, and M. Riedmiller, "Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards," *arXiv preprint arXiv:1707.08817*, 2017.

[4] L. Grinsztajn, E. Oyallon, and G. Varoquaux, "Why do tree-based models still outperform deep learning on typical tabular data?" *Advances in neural information processing systems*, vol. 35, pp. 507–520, 2022.

[5] Z. Xie, Z. Lin, J. Li, S. Li, and D. Ye, "Pretraining in deep reinforcement learning: A survey," *arXiv preprint arXiv:2211.03959*, 2022.

[6] L. Claussmann, M. Revilloud, D. Gruyer, and S. Glaser, "A review of motion planning for highway autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1826–1848, 2019.

[7] T. Shi, P. Wang, X. Cheng, C.-Y. Chan, and D. Huang, "Driving decision and control for automated lane change behavior based on deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019, pp. 2895–2900.

[8] C.-J. Hoel, K. Wolff, and L. Laine, "Automated speed and lane change decision making using deep reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2148–2155.

[9] B. Mirchevska, C. Pek, M. Werling, M. Althoff, and J. Boedecker, "High-level decision making for safe and reasonable autonomous lane changing using reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018, pp. 2156–2162.

[10] J. Wang, Q. Zhang, D. Zhao, and Y. Chen, "Lane change decision-making through deep reinforcement learning with rule-based constraints," in *International Joint Conference on Neural Networks (IJCNN)*, 2019, pp. 1–6.

[11] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dynamic Systems*, vol. 13, no. 1-2, pp. 41–77, 2003.

[12] J. Zhang, H. Yu, and W. Xu, "Hierarchical reinforcement learning by discovering intrinsic options," in *International Conference on Learning Representations (ICLR)*, 2020, pp. 1–19.

[13] J. Chen, Z. Wang, and M. Tomizuka, "Deep hierarchical reinforcement learning for autonomous driving with distinct behaviors," in *IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1239–1244.

[14] Z. Qiao, Z. Tyree, P. Mudalige, J. Schneider, and J. M. Dolan, "Hierarchical reinforcement learning method for autonomous vehicle behavior planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6084–6089.

[15] H. Wu, Y. Li, H. Zhuang, C. Wang, and M. Yang, "Hr-planner: A hierarchical highway tactical planner based on residual reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2022, pp. 7263–7269.

[16] J. Duan, S. Eben Li, Y. Guan, Q. Sun, and B. Cheng, "Hierarchical reinforcement learning for self-driving decision-making without reliance on labelled driving data," *IET Intelligent Transport Systems*, vol. 14, no. 5, pp. 297–305, 2020.

[17] Z. Qiao, J. Schneider, and J. M. Dolan, "Behavior planning at urban intersections through hierarchical reinforcement learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 2667–2673.

[18] J. Zhao, V. L. Knoop, J. Sun, Z. Ma, and M. Wang, "Unprotected left-turn behavior model capturing path variations at intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 9, pp. 9016–9030, 2023.

[19] Y. Rahmati, M. K. Hosseini, and A. Talebpour, "Helping automated vehicles with left-turn maneuvers: A game theory-based decision framework for conflicting maneuvers at intersections," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 8, pp. 11 877–11 890, 2022.

[20] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 961–971.

[21] J. Hong, B. Sapp, and J. Philbin, "Rules of the road: Predicting driving behavior with a convolutional model of semantic interactions," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 8446–8454.

[22] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, "Learning lane graph representations for motion forecasting," in *European Conference on Computer Vision (ECCV)*, 2020, pp. 541–556.

[23] J. Ngiam, V. Vasudevan, B. Caine, Z. Zhang, H. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. J. Weiss, B. Sapp, Z. Chen, and J. Shlens, "Scene transformer: A unified architecture for predicting future trajectories of multiple agents," in *International Conference on Learning Representations (ICLR)*, 2022, pp. 1–25.

[24] Z. Wang, J. Guo, Z. Hu, H. Zhang, J. Zhang, and J. Pu, "Lane transformer: A high-efficiency trajectory prediction model," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 4, pp. 2–13, 2023.

[25] Y. Chen, C. Dong, P. Palanisamy, P. Mudalige, K. Muelling, and J. M. Dolan, "Attention-based hierarchical deep reinforcement learning for lane change behaviors in autonomous driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3697–3703.

[26] K. B. Naveed, Z. Qiao, and J. M. Dolan, "Trajectory planning for autonomous vehicles using hierarchical reinforcement learning," in *IEEE International Conference on Intelligent Transportation Systems Conference (ITSC)*, 2021, pp. 601–606.

[27] J. Liu, X. Mao, Y. Fang, D. Zhu, and M. Q.-H. Meng, "A survey on deep-learning approaches for vehicle trajectory prediction in autonomous driving," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2021, pp. 978–985.

[28] C. J. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, pp. 279–292, 1992.

[29] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[30] B. Wilson, W. Qi, T. Agarwal, J. Lambert, J. Singh, S. Khandelwal, B. Pan, R. Kumar, A. Hartnett, J. K. Pontes, D. Ramanan, P. Carr, and J. Hays, "Argoverse 2: Next generation datasets for self-driving perception and forecasting," 2023.

[31] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on Robot Learning (CoRL)*, 2017, pp. 1–16.

[32] X. Wang, D. Zhao, H. Peng, and D. J. LeBlanc, "Analysis of unprotected intersection left-turn conflicts based on naturalistic driving data," in *IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 218–223.