GameFormer Planner: A Learning-enabled Interactive Prediction and Planning Framework for Autonomous Vehicles

Zhiyu Huang, Haochen Liu, Xiaoyu Mo, Chen Lyu
Nanyang Technological University, Singapore
zhiyu001@e.ntu.edu.sg, lyuchen@ntu.edu.sg

Abstract

*Decision-making is a fundamental yet challenging task for autonomous vehicles, as it requires accurate predictions of other traffic participants and, above all, safe and interactive plans for the ego vehicle. This study introduces the **GameFormer planner**, a planning framework that builds upon the GameFormer model. The framework consists of four key steps: feature processing, path planning, model query, and trajectory refinement. At the heart of our planning framework lies the GameFormer model, which is a Transformer-based neural network specifically designed for interactive prediction and planning. It generates an initial plan for the ego vehicle and predicts the trajectories of surrounding agents through iterative decoding refinements. Comprehensive evaluations conducted on the nuPlan benchmark demonstrate the competitive performance of the proposed planning framework, validating its effectiveness in both open-loop and closed-loop tests. The code is available at https://github.com/MCZhi/GameFormer-Planner/.*

1. Introduction

One of the critical challenges faced by autonomous vehicles (AVs) is the ability to make informed decisions in complex traffic scenarios within real-world environments. Effective decision-making, especially in situations involving interactions with other agents, relies on a comprehensive understanding of their intentions and accurate prediction of their future movements. To address this, numerous studies have leveraged machine learning models [5, 6, 8] to tackle the prediction problem. However, focusing solely on prediction accuracy is insufficient to guarantee optimal planning performance [4], given the intricate interconnections between the actions of the ego vehicle and other agents. Therefore, it becomes crucial to develop joint planning and prediction methods [2, 3] capable of reasoning about the interactions between agents and the ego vehicle. Such methods enable the generation of joint and coordinated decisions, enhancing the interactivity of the planning processes.

In this study, we present an extension of our previous work on interactive prediction and planning. Building upon the success of the GameFormer model [2], which has demonstrated state-of-the-art performance in the interaction prediction task, we introduce the **GameFormer planner** as a comprehensive planning framework. The planning process comprises the following sequential steps: 1) *feature processing*: relevant input data from the observation buffer and map undergoes preprocessing to extract pertinent features for the model; 2) *path planning*: potential reference paths for the ego vehicle are computed and the optimal path is selected; 3) *model query*: the GameFormer model is queried to generate an initial plan for the ego vehicle and predict the trajectories of surrounding agents; and 4) *trajectory refinement*: a nonlinear optimizer is employed to refine the ego vehicle's initial trajectory and produce the final plan.

2. Method

2.1. Planning Framework

The overall planning framework is illustrated in Figure 1, which consists of four key steps. Firstly, feature preprocessing involves extracting historical trajectories of surrounding agents from the simulation buffer and obtaining vectorized map polylines from the map API. Next, path planning utilizes a state lattice planner to sample multiple waypoints along the route centerlines, connecting the ego vehicle's current position and target waypoints using Bezier curves. An optimal path is then selected based on a cost function. In the model query step, the GameFormer model is employed to generate an initial plan for the ego vehicle based on environmental inputs and to predict trajectories for surrounding agents. Lastly, trajectory refinement employs a nonlinear optimizer to refine the initial plan, considering factors such as maintaining target speed, minimizing position error, jerk, and acceleration, as well as ensuring collision avoidance. The GameFormer model is trained on the nuPlan dataset, and the planning framework is evaluated using the accom-

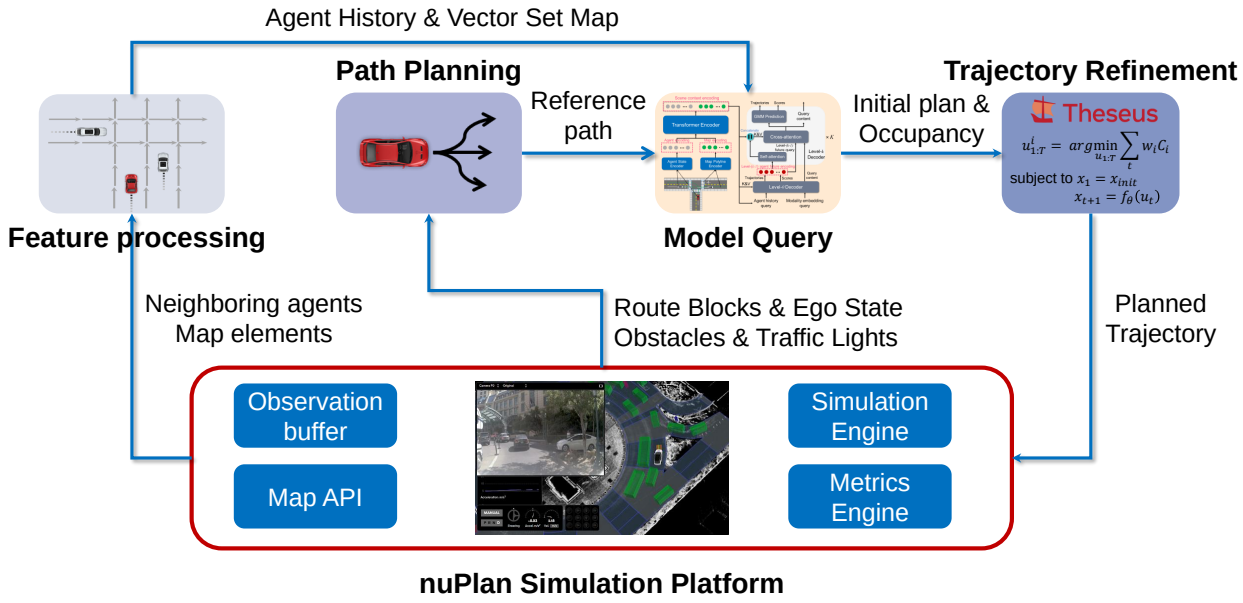

Figure 1. An overview of the GameFormer planner. The framework comprises four key steps: feature processing, path planning, model querying, and trajectory refinement. These steps extract information from the simulation platform and generate planned trajectories.

panying simulation platform and benchmark [1]. Further details of each step in the framework will be provided in the following subsections.

2.2. Feature Processing

The GameFormer model requires two distinct types of feature inputs: the historical trajectories of agents and a vectorized map. In contrast to the original configuration in GameFormer, where we find possible routes and crosswalks for individual agents, we now employ a global vector set map structure that encompasses various map elements in close proximity to the ego vehicle. This map information is shared among all agents.

Regarding the surrounding agents, we find the $N = 20$ agents that are closest to the ego vehicle at the current timestep, encompassing various agent types (i.e., vehicles, pedestrians, and cyclists). To capture their historical behavior, we sample the timesteps from the observation buffer at a frequency of 10 Hz, covering the past 2 seconds along with the current timestep, resulting in a total of $T_h = 21$ timesteps. Consequently, we obtain a feature tensor that represents the historical information of neighboring agents, with a shape of $N \times T_h \times 11$. The features encompass 11 dimensions, including diverse attributes of the agents such as their x and y coordinates, heading angle h, velocities v_x and v_y, yaw rate γ, size (length and width), and one-hot encoding of agent type. Furthermore, we extract the historical states of the ego vehicle and organize them into a tensor with a shape of $1 \times T_h \times 7$. This tensor incorporates features of the ego vehicle's pose (x, y, h), velocities (v_x, v_y), and accelerations (a_x, a_y). It is important to note that all positional and velocity attributes are normalized relative to the ego vehicle's current state, and any vacant positions within the tensors are padded with zeros.

For the vectorized map, we extract three specific types of map elements (i.e., lanes, crosswalks, and route lanes) from the map API. To mitigate potential timeout errors on the evaluation server, we set the search radius to $r = 10$ meters. However, we highly recommend utilizing a larger radius of $r \geq 50$ meters to achieve better performance. The map lane tensor has a shape of $40 \times 50 \times 7$, allowing for a maximum of 40 lanes, each consisting of 50 centerline waypoints. The features within the map lane tensor encompass 7 dimensions, including the position (x, y, h) of each waypoint and the one-hot encoding of the traffic light state. In cases where there are insufficient elements, we pad the positions with zeros. Similarly, the map crosswalk tensor has a shape of $5 \times 30 \times 3$, and the map route lane tensor has a shape of $10 \times 50 \times 3$. The positional attributes of these tensors are normalized relative to the ego vehicle's state.

2.3. Path Planning

To ensure closed-loop planning performance, it is necessary to have a reference path that guides and constrains the planned trajectories within the lane while adhering to traffic regulations. The path planning process unfolds as follows.

1) Finding possible route plans: Initially, candidate route edges around the current position of the ego vehicle are identified, followed by the utilization of a depth-first search algorithm to find candidate route plans within the designated route blocks. Once the route plans are obtained, the corresponding centerlines are extracted, resulting in multiple path options. A valid path typically extends approximately 120 to 150 meters, and any route paths with lengths

below a predefined threshold are disregarded.

2) Generating candidate paths: We employ a state lattice planner to generate the candidate paths. Initially, multiple waypoints are sampled along each route path at fixed intervals of $[5, 10, 15, 20]$ meters, with the origin starting from the position closest to the ego vehicle. These target waypoints on all route paths collectively form a state lattice. Subsequently, each target state is connected to the ego vehicle's current position using Bezier curves, resulting in a smooth path. The remaining segment of the candidate path follows the centerline of the target route path. By applying this process to all target states on the lattice, multiple candidate paths can be generated. The path generation process is illustrated in Figure 2.

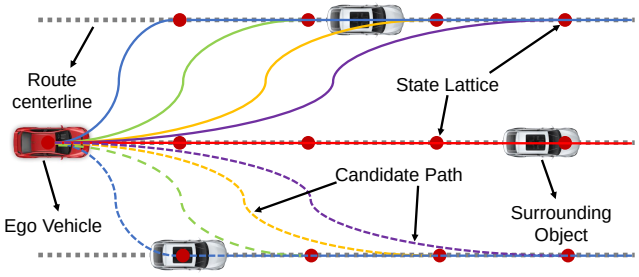

Figure 2. Illustration of path generation process

3) Selecting the optimal path: After generating the candidate paths, a selection process is employed to determine the optimal path. Several criteria are taken into account, such as path curvature, presence of obstacles, lane changes, crossing road boundaries, and reaching the target lane. A cost function is utilized to compute the cost associated with each path, and the path with the lowest cost is selected. More specifically, the cost of a path ζ is determined by a weighted sum of different features: $c(\zeta) = \sum_i w_i f_i(\zeta)$. These features encompass the maximum curvature of the path (f_1), the distance of the path to the ego vehicle (f_2), a binary indicator representing whether the path intersects with any static obstacle (f_3), a binary indicator indicating whether the path intersects with any road boundaries (f_4), and a binary indicator signifying the allowance to change to the target lane (the longest route path) (f_5). The weights assigned to these features are as follows: $w_1 = 0.1, w_2 = 1, w_3 = 10, w_4 = 2, w_5 = -5$.

4) Post-processing: After selecting the optimal path, post-processing steps are performed to refine and finalize the reference path. We utilize cubic spline interpolation to smooth the path and calculate the curvature of the path. Furthermore, we adjust the coordinates and headings of the path to be relative to the ego vehicle's position. The waypoints along the path are uniformly spaced at intervals of 0.1 meters, with a maximum path length constraint of 120 meters. Subsequently, we annotate the target speed based on the speed limit and path curvature, as well as the occupancy resulting from traffic lights encountered along the path. Finally, we obtain a tensor of the reference path with a shape of 1200×6, encompassing the waypoint coordinates (x, y), heading, curvature, target speed, and occupancy.

2.4. Model Query

To ensure efficient inference speed, we utilize a compact version of the GameFormer model [2]. This model is composed of 3 encoding layers and 3 decoding layers, which include 1 initial decoding layer and 2 interaction decoding layers. Additionally, we introduce an extra decoding layer following the last interaction decoding layer to generate the ego vehicle's plan. The decoding layer is implemented as a Transformer module, where the query is derived from the max-pooled latent feature of the ego vehicle along the modality dimension, and the key and value are obtained by concatenating the latent feature with the encoded route lanes. The output of this layer is a tensor with a shape of $1 \times 80 \times 2$, representing the ego plan with future x and y coordinates. The ego plan is then projected onto the reference path as an initialization of the refine planner.

The output of the GameFormer model consists of multimodal trajectories for the surrounding $N = 20$ agents, represented by tensors of shape $N \times 6 \times 80 \times 2$. The model also provides probabilities associated with the different modes of each agent's trajectory, resulting in tensors of shape $N \times 6$. For each agent, we select the trajectory with the highest probability and project it onto the reference path using the Frenet transformation [9] to calculate the spatial-temporal occupancy. Specifically, we initialize the occupancy grid O as a zero tensor of shape $80 \times 120 \times 1$. For any agent at any given timestep, if the projected lateral coordinate $d \leq 0.5(W_e + W_a) + 0.5$, where W_e is the width of the ego vehicle and W_a is the width of the agent, and the projected longitudinal coordinate $0 < s < 120$, we annotate the path occupancy for that timestep. The occupied area along the path is defined as $[s - 0.5L_a + 3, s + 0.5L_a]$, where L_a is the length of the agent. The final occupancy grid is obtained by combining the traffic light occupancy using the maximum operation. An example illustrating the derivation of the occupancy grid is shown in Figure 3.

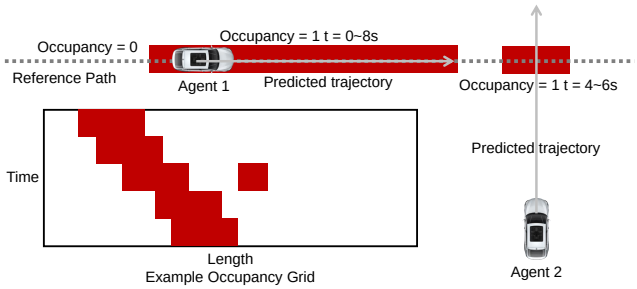

Figure 3. Illustration of occupancy grid calculation

2.5. Trajectory Refinement

ML-based policies often face challenges related to distributional shifts and causal confusion, which can result in poor performance in closed-loop tests. To mitigate these issues, we employ a nonlinear optimizer [7] as a refine motion planner within our framework. The refine planner plays a crucial role in explicitly regulating the comfort and speed of the trajectory while ensuring collision avoidance. Specifically, the refine planner optimizes the speed along the reference path $\dot{s}_t$, utilizing the following objective function:

$$\dot{s}_t^* = \arg\min_{\dot{s}_t} \sum_i \sum_t \omega_i c_i(\dot{s}_t), \tag{1}$$

where ω_i is the weight assigned to each individual cost term c_i, and t denotes the time index.

The individual cost terms, $c_i(\dot{s}_t)$, capture various factors such as comfort, safety, and travel efficiency. We begin by considering the speed cost:

$$c_{\text{speed}} = \| \dot{s}_t - v_{\text{target}}(\hat{s}) \|^2, \tag{2}$$

where v_{target} is the target speed along the reference path, and $\hat{s}$ is the initial plan provided by the model. Additionally, we introduce a soft constraint to prevent negative speeds:

$$c_{\text{forward}} = \begin{cases} 0 & \text{if } \dot{s}_t \geq 0, \\ \| \dot{s}_t \|^2 & \text{otherwise.} \end{cases} \tag{3}$$

Next, we consider ride comfort factors, including acceleration and jerk:

$$c_{\text{acc}} = \| \ddot{s}_t \|^2, \tag{4}$$

$$c_{\text{jerk}} = \| \dddot{s}_t \|^2 . \tag{5}$$

To ensure comfort, we introduce a soft constraint on the acceleration, which should be within a specified range:

$$c_{\text{comfort}} = \begin{cases} 0 & \text{if } -4.05 \leq \ddot{s}_t \leq 2.4, \\ \| \ddot{s}_t \|^2 & \text{otherwise.} \end{cases} \tag{6}$$

We also include a cost term to encourage the final step of the refined trajectory to be close to the initial plan $\hat{s}$:

$$c_{\text{init}} = \begin{cases} 0 & \text{if } \| s_{T_f} - \hat{s}_{T_f} \| \leq 3, \\ \| s_{T_f} - \hat{s}_{T_f} \|^2 & \text{otherwise,} \end{cases} \tag{7}$$

where s is the coordinate on the reference path obtained by integrating the speed $\dot{s}_t$, and T_f is the planning horizon.

Finally, we address collision avoidance by ensuring that the vehicle's position does not exceed occupied areas.

$$c_{\text{collision}} = \begin{cases} \| s_t + L_e - O_t \|^2 & \text{if } O_t > \hat{s}_t,\ s_t + L_e > O_t, \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

where L_e is the length of the ego vehicle, and O_t is the occupancy grid at timestep t. The condition $O_t > \hat{s}_t$ indicates that we only consider the occupancy grid ahead of the position in the initial plan. To improve runtime, we focus on specific timesteps at the beginning of the trajectory, $t = [0.2, 0.4, 0.7, 1.0, 1.5, 2.0, 2.5, 3]\ s$.

The optimization problem is solved using the Gauss-Newton method with the following settings: a maximum iteration of 10, a step size of 0.5, and a relative error tolerance of 1e-3. The cost terms are assigned the following weights: $\omega_{\text{speed}} = 0.3, \omega_{\text{acc}} = 0.1, \omega_{\text{jerk}} = 1.0, \omega_{\text{init}} = 3.0$, and the soft constraints are assigned with a weight of 10. While assuming the lateral movement is $d_t = 0$, for future work aiming to achieve a more flexible plan, we will consider optimizing d or jointly optimizing both s and d. Once the refined trajectory s^* is obtained, we project the coordinates back to Cartesian space, resulting in the final planned trajectory represented as a sequence of (x, y, h).

2.6. Model Training

The GameFormer model is trained using the train and validation partitions of the nuPlan dataset, which have undergone data preprocessing and contain approximately 2.8 million data points. We employ GMM loss on the best joint modal for predicting agent behaviors and smooth L1 loss for the ego plan. In consideration of speed, the original collision loss is not utilized. The training process is performed on 4 NVIDIA Tesla A100 GPUs, with a batch size of 256 on each GPU. The AdamW optimizer is employed with an initial learning rate of 1e-4 and weight decay of 1e-2. The learning rate is decreased by a factor of 0.5 every 3 epochs, starting from the 10th epoch. The training is carried out for a total of 30 epochs.

3. Results

The evaluation of our planning framework is conducted on the nuPlan benchmark, including three tasks: open-loop (OL) planning, closed-loop (CL) planning with non-reactive agents, and closed-loop planning with reactive agents. A variety of metrics are employed to calculate the overall score for each task. Table 1 presents the results of our planning framework compared to other competing methods. The results demonstrate the effectiveness and capability of our planning framework in achieving high-quality planning results across the evaluated tasks.

Table 1. Testing results on the nuPlan planning benchmark

Method	OL	CL non-reactive	CL reactive
hoplan	0.87	0.89	0.88
pegasus_multi_path	0.88	0.82	0.85
GameFormer (Ours)	0.84	0.81	0.84
ltp-planner	0.88	0.79	0.80
Forecast_MAE	0.91	0.76	0.73
UrbanDriver	0.86	0.68	0.70
IDMPlanner	0.29	0.72	0.75

References

[1] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles. In *CVPR ADP3 workshop*, 2021. 2

[2] Zhiyu Huang, Haochen Liu, and Chen Lv. Gameformer: Game-theoretic modeling and learning of transformer-based interactive prediction and planning for autonomous driving. *arXiv preprint arXiv:2303.05760*, 2023. 1, 3

[3] Zhiyu Huang, Haochen Liu, Jingda Wu, and Chen Lv. Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving. *arXiv preprint arXiv:2207.10422*, 2022. 1

[4] Zhiyu Huang, Haochen Liu, Jingda Wu, and Chen Lv. Conditional predictive behavior planning with inverse reinforcement learning for human-like autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 2023. 1

[5] Zhiyu Huang, Xiaoyu Mo, and Chen Lv. Multi-modal motion prediction with transformer-based neural network for autonomous driving. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2605–2611. IEEE, 2022. 1

[6] Xiaoyu Mo, Zhiyu Huang, Yang Xing, and Chen Lv. Multi-agent trajectory prediction with heterogeneous edge-enhanced graph attention network. *IEEE Transactions on Intelligent Transportation Systems*, 2022. 1

[7] Luis Pineda, Taosha Fan, Maurizio Monge, Shobha Venkataraman, Paloma Sodhi, Ricky TQ Chen, Joseph Ortiz, Daniel DeTone, Austin Wang, Stuart Anderson, et al. Theseus: A library for differentiable nonlinear optimization. *Advances in Neural Information Processing Systems*, 35:3801–3818, 2022. 4

[8] Shaoshuai Shi, Li Jiang, Dengxin Dai, and Bernt Schiele. Motion transformer with global intention localization and local movement refinement. In *Advances in Neural Information Processing Systems*, 2022. 1

[9] Moritz Werling, Julius Ziegler, Sören Kammel, and Sebastian Thrun. Optimal trajectory generation for dynamic street scenarios in a frenét frame. In *2010 IEEE International Conference on Robotics and Automation*, pages 987–993, 2010. 3