IEEE Robotics and Automation Letters (RA-L) paper, presented at ICRA 2026, Vienna, Austria. Cite as RA-L paper.

KiGRAS: Kinematic-Driven Generative Model for Realistic Agent Simulation

Jianbo Zhao[1,2,*], Jiaheng Zhuang[2,3,*], Qibin Zhou[2,*], Taiyu Ban[4,*], Ziyao Xu[2,†], Hangning Zhou[2,†], Junhe Wang[2], Guoan Wang[2], Zhiheng Li[3], Bin Li[1]

***Abstract*—Trajectory generation is a pivotal task in autonomous driving. Recent studies have introduced the autoregressive paradigm, leveraging the state transition model to approximate future trajectory distributions. This paradigm closely mirrors the real-world trajectory generation process and has achieved notable success. However, its potential is limited by the ineffective representation of realistic trajectories within the redundant state space. To address this limitation, we propose the Kinematic-Driven Generative Model for Realistic Agent Simulation (KiGRAS). Instead of modeling in the state space, KiGRAS factorizes the driving scene into action probability distributions at each time step, providing a compact space to represent realistic driving patterns. By establishing physical causality from actions (cause) to trajectories (effect) through the kinematic model, KiGRAS eliminates massive redundant trajectories. All states derived from actions in the causal space are constrained to be physically feasible. Furthermore, redundant trajectories representing identical action sequences are mapped to the same representation, reflecting their underlying actions. This approach significantly reduces task complexity and ensures physical feasibility. KiGRAS achieves state-of-the-art performance in Waymo's SimAgents Challenge, ranking first on the WOMD leaderboard with significantly fewer parameters than other models.**

***Index Terms*—Deep Learning Methods, Trajectory Generation, Motion Prediction, Agents Simulation.**

I. Introduction

SMART autonomous driving (AD) systems depend significantly on the generation of realistic future trajectories for agents, which is crucial for tasks such as motion prediction [8, 34], motion planning [6, 7, 31], and agent simulation [15]. In the context of trajectory distribution modeling, physical feasibility and interaction fidelity are two essential premises for realistic trajectory generation. Physical feasibility [14] ensures that trajectories comply with the fundamental principles of vehicle dynamics and kinematics. Interaction fidelity, on the other hand, ensures realism and consistency in modeling the interactions between agents. Adequate capture of these two aspects remains a major challenge in current AD research.

Previous studies [8, 34] employ mainly a direct regressive paradigm to approximate future trajectory distributions using Gaussian or Laplace distributions. However, a finite number of these distributions struggle to accurately model complex realistic trajectories, fundamentally limiting the performance of these models. Moreover, the regressive methods inadequately capture agent interactions in future trajectories [20], compromising the interaction fidelity of multi-agent trajectories.

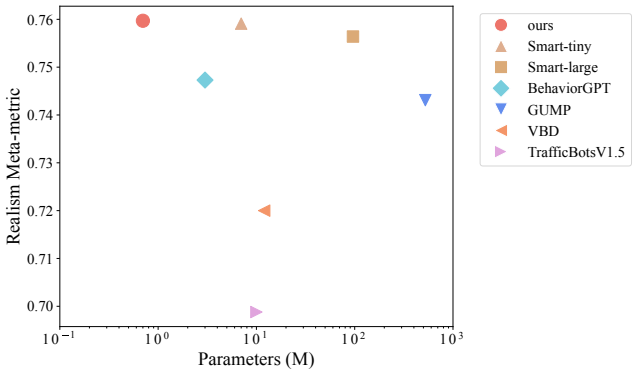


Fig. 1. Performance comparison of different models based on Parameters (M) and Realism Meta-metric, the overall metric of Waymo's SimAgent. Each marker represents a specific model with its respective parameter size and realism meta score. For more details, see Section IV-C.

Inspired by the success of autoregressive language models [1], recent studies have proposed an autoregressive paradigm for trajectory generation [16, 20]. Unlike previous direct regression in a long future period, these methods model agent and road data as a sequence of short-period state tokens, and process autoregression on these tokens, predicting a short future of trajectories at each step. This autoregressive paradigm better models the complexity of trajectory distributions and more accurately captures the interactions between agents.

However, existing autoregressive methods model directly in the trajectory state space, which is limited by two types of redundancies [22]. First, realistic trajectories constitute only a small subset of the predictive space, and there is a significant presence of trajectories that violate physical laws. This makes the state space *inefficient* for representing physically feasible trajectories, increasing training difficulty and challenging the physical feasibility of trajectory generation. Second, different agents exhibit similar kinematic patterns. When modeling state transitions in the state space, a significant amount of information entropy is repeatedly used to describe the same kinematic state transition model. This redundant representation adds *unnecessary* complexity, limiting the effectiveness and potential of the autoregressive paradigm.

To address these challenges, we propose Kinematic-driven Generative Model for Realistic Agent Simulation (KiGRAS). KiGRAS presents a new autoregressive paradigm by reformulating the task from trajectory distribution modeling to

* These authors contributed equally to this work.
† Corresponding authors: Ziyao Xu; Hangning Zhou. (e-mail: ziyao.xu@mach-drive.com; hangning.zhou@mach-drive.com).
[1] School of Information Science and Technology, University of Science and Technology of China, 96 Jinzhai Rd, Hefei 230026, China.
[2] Mach Drive, Ruixiang Road 88, Wuhu, China.
[3] Tsinghua Shenzhen International Graduate School, Tsinghua University, 30 Shuangqing Rd, Haidian District, Beijing 100084, China.
[4] School of Computer Science and Technology, University of Science and Technology of China, 96 Jinzhai Rd, Hefei 230026, China.

©2026 IEEE

action distribution modeling. We introduce the kinematic model to inversely infer control actions from trajectories of consecutive time steps, thus establishing physical causality from the control action (cause) to trajectories (effect). Instead of existing paradigms learning in the *effect space*, KiGRAS predicts future control actions, which learns in the *causal space* that captures a more fundamental aspect of driving patterns. In the action space, the massive trajectory redundancies are transformed into concise action sequences, creating a compact representation of driving patterns and greatly reducing the complexity of learning realistic driving behaviors. Moreover, KiGRAS naturally ensures the physical feasibility of trajectories in all time steps by a kinematic-driven forward update mechanism. Prior to inference, the state of all agents is updated using the previous state and the predicted control action through a forward kinematic calculation. This kinematic-driven forward update of agent states builds a hard constraint on state transition, making the state of each time step reachable by control. Our contributions are listed threefold:

- We propose KiGRAS, a novel autoregressive paradigm for the trajectory generation task that learns in the control action space, which makes a compact representation of driving patterns and fundamentally reduces the complexity of realistic trajectory generation.
- We propose a kinematic-driven approach for inverse inference of control actions and forward update of agent states, building physical causality from control actions to trajectories and naturally ensuring the physical feasibility of each time step of generated trajectories.
- We propose a unified network architecture, which unifies the task definition at each time step and unifies scene representation with the same spatial encoder. This unification further simplifies the task complexity, enhances agent interaction modeling, and introduces ease of post fine-tuning for more fine-grained customization.

KiGRAS (0.7M) achieves state-of-the-art performance in the Sim Agents challenge[1], ranking **1st** on the WOMD leaderboard with significantly fewer parameters than other models, as reported in Fig. 1. The reduced parameter count offers lower model memory and computational costs, making deployment feasible in resource-limited environments. Interestingly, KiGRAS can embed all human preferences present in the data, similar to InstructGPT [1], allowing for the customization of different driving habits. We provide a fine-tuning approach for tailoring driving preferences. For more details, see Section III-E.

II. Related work

A. Regressive trajectory Generation

Predominant deep learning (DL) methods are based on the regressive paradigm to model trajectory distributions in a long future period [11]. In the early stage, some studies [8, 34] directly take historical trajectories as input and future trajectories as labels, and use the regression loss to supervise the training process. These methods assumed that the future trajectory space followed a single Gaussian distribution, which is too poor to model realistic cases. Subsequently, some studies [35, 36] generated multimodal trajectories and employed probabilistic regression models to capture the trajectory generation process. These researchers assumed that the future trajectory probability space could be approximated by the superposition of multiple Gaussian [8] or Laplace [34] distributions. Yet in practice, finite Gaussian or Laplace distributions are still not sufficient enough to describe the complex realistic distributions of trajectories, especially for long-term future trajectories. Additionally, VBD and related approaches [13, 27, 30] employs diffusion models or generative adversarial networks to address trajectory generation by modeling the distribution of future trajectories. However, the complexity of long-term trajectory prediction limits performance.

B. Autoregressive trajectory generation

Inspired by the success of autoregressive models [4, 24] in the language field, a sequential prediction task sharing many similarities to trajectory generation, several studies have used this autoregressive paradigm to model trajectory distributions, reducing the long future prediction to a step-by-step prediction of the short future period [16]. Seff $et\ al.$ introduce MotionLM [20], which tokenizes trajectories into actions, encodes historical environments and predicts the next action using the previous action during the decoding process. However, these approaches do not update the environment in real-time during decoding, limiting performance in complex scenarios.

Building on these approaches, BehaviorGPT [33], MVTE [26], and TrafficBOTv1.5 [32] use regression models to iteratively predict potential trajectory states, updating environmental or action information after each prediction and re-encoding it for subsequent steps. Similarly, SMART [29] and GUMP [12] tokenize continuous trajectory spaces and employ classification loss to guide training. At each time step, they predict the probability distribution over the trajectory space for the subsequent state, selecting trajectory segments based on specific sampling strategies.

A major difference between our KiGRAS and these methods is that we fundamentally redefine the task in the autoregressive paradigm of trajectory generation. Instead of modeling trajectory distributions filled with massive redundant instances stemming from the same control action, we directly model the space of control actions. This re-formulation provides a much more compact predictive space of realistic driving patterns, thereby significantly reducing the task complexity. Additionally, KiGRAS inherently ensures the physical feasibility of generated trajectories at each time step by imposing a hard constraint on control reachability between consecutive steps, a criterion not satisfied by the above methods.

III. Kinematic-Driven Generative Model

This section introduces details of our KiGRAS. We start with the problem definition of traditional trajectory generation methods, and then introduce our re-formulation to control action modeling. Following this, we illustrate details of the control action representation and the kinematic-driven autoregression paradigm. Finally, we introduce a fine-tuning approach in post-training for driving habit customization.

[1]https://waymo.com/open/challenges/2024/sim-agents/

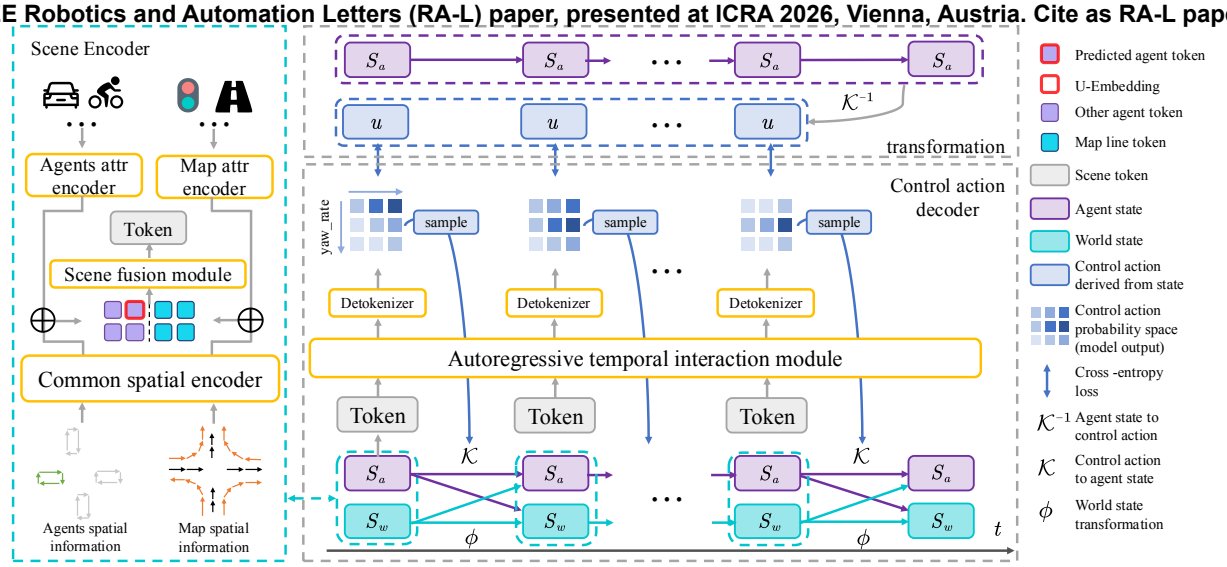


Fig. 2. The architecture of the KiGRAS framework. First, we solve for the sequence of control actions from trajectory states using the inverse kinematic transformation module. These actions are then represented in a discrete space for processing the forward update of the inference process. To encode the traffic scene, we use a common spatial encoder to embed the spatial information of all agents and map lines, along with two attribute encoders to describe their traffic roles. Building on this, an autoregressive transform-based decoder is designed to decode the action probability distributions for each state.

A. Problem definition

Given the historical states from time $-n$ to time 0 (current time) of one or multiple target agents, $\{\mathcal{S}^{\mathrm{a}}_{-n}, \ldots, \mathcal{S}^{\mathrm{a}}_{0}\}$, and the surrounding world environment $\{\mathcal{S}^{\mathrm{w}}_{-n}, \ldots, \mathcal{S}^{\mathrm{w}}_{0}\}$, the goal is to generate the future states of the target agent from time 1 to T, denoted as $\{\mathcal{S}^{\mathrm{a}}_{1}, \mathcal{S}^{\mathrm{a}}_{2}, \ldots, \mathcal{S}^{\mathrm{a}}_{T}\}$. For brevity, we denote the set of target agent's future states from time 1 to time T as $\mathcal{S}^{\mathrm{a}}_{1:T}$ and the historical states as $\mathcal{S}^{\mathrm{a}}_{\leq 0}$.

The target agent, denoted as $\mathcal{S}^{\mathrm{a}}$, includes parameters such as position, heading, and velocity, while $\mathcal{S}^{\mathrm{w}}$ represents the world environment state, including map data (S^{map}) and the states of other agents (S^{oa}). Here, S^{oa} encompasses the states of all other entities besides the target agent, including motor vehicles, non-motorized vehicles, pedestrians and other obstacles:

$$\mathcal{S}^{\mathrm{w}} = \{S^{\mathrm{map}}, S^{\mathrm{oa}}\} \tag{1}$$

This task is formalized to maximize the likelihood of the target agents' future state distribution given the historical states:

$$\arg\max_{\theta} P_{\theta}(\mathcal{S}^{\mathrm{a}}_{1:T} \mid \mathcal{S}^{\mathrm{w}}_{\leq 0}, \mathcal{S}^{\mathrm{a}}_{\leq 0}) \tag{2}$$

Typically, the future time horizon T spans several seconds, which is relatively long in the context of a driving scenario. Consequently, this task entails a long-term temporal prediction based on historical data.

B. Re-formulation to control action modeling

Probability factorization. The trajectory distribution formulation in Eq. (2) can be decomposed into a product of conditional distributions over each time step.

$$P(\mathcal{S}^{\mathrm{a}}_{1:T} \mid \mathcal{S}^{\mathrm{w}}_{\leq 0}, \mathcal{S}^{\mathrm{a}}_{\leq 0}) = \prod_{t=0}^{T-1} P(\mathcal{S}^{\mathrm{a}}_{t+1} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}) \tag{3}$$

This decomposition transforms the complex task of long-term trajectory prediction into a series of next-step prediction tasks. Each next-step prediction is significantly less difficult than solving Equation (2), as the goal is reduced to predicting the immediate next state based on the history up to that point, rather than tackling the entire sequence at once.

Training objective shift We further delve into the next-step prediction task and introduce the variable of control action U to enable more fine-grained modeling of the transitions between temporal trajectory states, formalized as:

$$S^{\mathrm{a}}_{t+1} = \mathcal{K}(S^{\mathrm{a}}_{t}, U^{\mathrm{a}}_{t}) \tag{4}$$

Here, $\mathcal{K}$ represents the deterministic kinematic model [21]. This model governs the transition from the current state S^{a}_{t} to the next S^{a}_{t+1} within U^{a}_{t}. Notably, Eq. (4) implies that given S^{a}_{t} and U^{a}_{t}, the next state S^{a}_{t+1} is independent on any other variables. Based on this, we re-formalize each factor on the right-hand side of Eq. (3) as follows:

$$\begin{aligned} & P\left(S^{\mathrm{a}}_{t+1} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}\right) \\ &= \int P\left(S^{\mathrm{a}}_{t+1} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, S^{\mathrm{a}}_{\leq t}, U^{\mathrm{a}}_{t}\right) \\ &\quad \times P\left(U^{\mathrm{a}}_{t} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}\right) dU^{\mathrm{a}}_{t} \\ &= \int P\left(S^{\mathrm{a}}_{t+1} \mid S^{\mathrm{a}}_{t}, U^{\mathrm{a}}_{t}\right) \\ &\quad \times P\left(U^{\mathrm{a}}_{t} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}\right) dU^{\mathrm{a}}_{t} \end{aligned} \tag{5}$$

The first equality stems from the law of total probability, and the second equality holds by combining Eq. (4) and the fact that $\{\mathcal{S}^{\mathrm{a}}_{t}\} \subseteq \{\mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}\}$. Furthermore, adhering to the constraint of Eq. (4), we have that $P\left(S^{\mathrm{a}}_{t+1} \mid S^{\mathrm{a}}_{t}, U^{\mathrm{a}}_{t}\right) = 1$. Combining this result with Eqs. (2)-(5), we transform the long-term trajectory state prediction in Eq. (2) into the next-step prediction task of control actions:

$$\begin{aligned} & \arg\max_{\theta} P_{\theta}(\mathcal{S}^{\mathrm{a}}_{1:T} \mid \mathcal{S}^{\mathrm{w}}_{\leq 0}, \mathcal{S}^{\mathrm{a}}_{\leq 0}) \\ \Longleftrightarrow \quad & \arg\max_{\theta'} \prod_{t=0}^{T-1} P_{\theta'}\left(U^{\mathrm{a}}_{t} \mid \mathcal{S}^{\mathrm{w}}_{\leq t}, \mathcal{S}^{\mathrm{a}}_{\leq t}\right) \\ & \text{subject to} \quad S^{\mathrm{a}}_{\tau+1} = \mathcal{K}(S^{\mathrm{a}}_{\tau}, U^{\mathrm{a}}_{\tau}) \\ & t \in \{0, 1, \cdots, T-1\},\ \tau \in \{0, 1, \cdots, t\} \end{aligned} \tag{6}$$

where θ and θ' represent the learnable model parameters. In summary, our re-formalization[2] is built upon the hard kinematic constraint $S^{\mathrm{a}}_{\tau+1} = \mathcal{K}(S^{\mathrm{a}}_{\tau}, U^{\mathrm{a}}_{\tau})$, which shifts the training objective from trajectories to control actions. Our world transition process ensures temporal interaction consistency among all agents. For readability, we omit the transition of S^{w} here; details can be found in Eq. (8). This new paradigm reduces the prediction space from complex trajectory distributions to a significantly simpler set of discrete control actions. Notably, this simplification does not result in any loss of data information. On the contrary, it ensures that all future trajectory states remain physically consistent, thereby improving trajectory modeling and reducing the prediction space.

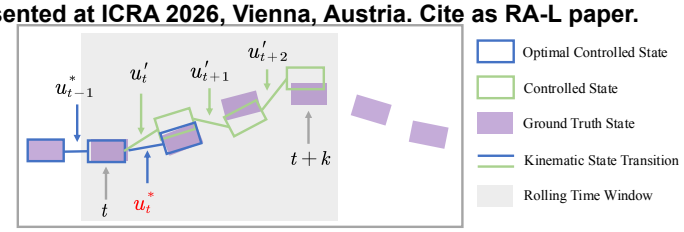


Fig. 3. An illustration of the Rolling Horizon Strategy in the inverse kinematic transformation module. We use the k consecutive states covered by the rolling time window to simultaneously optimize the sequence of control actions for these states (green-lined blocks). The optimal action for the closest future state (blue-lined blocks) is taken as output to mitigate accumulated errors. After these steps, the rolling time window moves forward one step to iteratively solve for all actions.

C. Discrete representation of control action

We represent control actions U in a discrete space. Specifically, the probability distribution of a control action $P(U)$ is defined as the joint probability $P(A, Y)$, with A representing acceleration and Y representing yaw rate. To represent realistic control actions, we utilize acceleration rate A in the range $[-5, 5]$ m/s^2 and the yaw rate Y in the range $[-1.5, 1.5]$ rad/s, which generally covers the control actions in the normal driving behavior context. On this basis, we make a fine-grained split of them into 63 bins. This results in a discrete action space U consisting of totally 63×63 control actions, which is flexible enough to model various detailed actions by agents, including vehicles or others.

Compared to previous regressive methods that employ continuous probability distributions, our discrete representation is not dependent on the assumption that probability distribution follows a superposition of multiple prior (Gaussian or Laplace) distributions, thus breaking the limitation of finite prior distributions that are usually too poor to approximate the realistic distributions.

D. Kinematic-driven autoregression

This section illustrates the details of the kinematic-driven autoregression process in KiGRAS. We first introduce inverse kinematic transformation module, which infers control actions from trajectory data to generate control actions as training labels. Subsequently, we describe the details of our model architecture and training data flow. Finally, we introduce the inference process.

Inverse kinematic transformation module To derive control actions between consecutive states, we employ the constant turn rate and acceleration model [2], which is a widely used approach for describing vehicle dynamics, to describe the physical relationship between the state S and the control action U. The definition of the control action U has been previously introduced in Section III-C. The agent state is represented as $s = (x, y, \theta, v)$, which comprises the pose (x, y, θ) and the velocity v. The continuous-time differential equations that govern the evolution of the state are expressed as follows:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega, \quad \dot{v} = a$$

where ω denotes the turn rate, and a represents the acceleration. It is important to note that the definitions of the agent states s, control actions u, and the kinematic model $\mathcal{K}$ can be substituted with alternative formulations that appropriately describe the kinematic behavior of the system.

To extract the discrete control action $U_{0:T-1}$ from the trajectory states $\mathcal{S}$, we integrate the Model Predictive Controller (MPC) method [10] into the transformation process. Concretely, we use the rolling horizon strategy to solve the discrete control sequence $\{u^*_0, u^*_1, \ldots, u^*_{T-1}\}$, as shown in Fig. 3. The process of solving for u^*_t at each rolling time window can be formulated as follows:

$$\begin{aligned} \min_{u_{t:t+k}} \sum_{\tau=t}^{t+k} \quad & \|s^{\mathrm{ctl}}_{\tau+1} - s_{\tau+1}\| \\ \text{subject to} \quad & s^{\mathrm{ctl}}_{\tau+1} = \mathcal{K}(s_\tau, u_\tau) \end{aligned} \tag{7}$$

To solve this problem, we use the state $s^{\mathrm{ctl}*}_t$, obtained by applying u^*_{t-1} from the previous rolling step, as the initial state for the current rolling time window. Subsequently, we use k continuous control parameters $\{u'_t, u'_{t+1}, \ldots, u'_{t+k-1}\}$ in conjunction with the kinematic model $\mathcal{K}$ to determine the control states $\{s^{\mathrm{ctl}}_{t+1}, \ldots, s^{\mathrm{ctl}}_{t+k}\}$ within the k-step time window [3]. We then optimize $\{u'_t, u'_{t+1}, \ldots, u'_{t+k-1}\}$.

Next, we find the discrete control action in the space U that is closest to the optimized continuous control action u'^*_t and designate it as u^*_t. Then, we apply the kinematic model $\mathcal{K}$ to transition the state $s^{\mathrm{ctl}*}_t$ to $s^{\mathrm{ctl}*}_{t+1}$ using u^*_t. The resulting state $s^{\mathrm{ctl}*}_{t+1}$ serves as the initial state for the next rolling step.

Model architecture In Fig. 2, the KiGRAS architecture consists of two main components: a scene encoder and a control action decoder. The scene encoder encodes multimodal features of agents and map lines at each time step into a scene token in a unified manner. The control action decoder then interacts with these scene tokens across the temporal sequence to decode the probability distribution of the control action space U at each time step. It then computes the cross-entropy loss using action labels from the inverse kinematic transformation module.

Scene encoder At each time step t, we use the same scene encoder to extract features from the scene information (Fig. 2 left). This includes the state of the predicted agent, the states of other agents, map information centered around the

[2]This formulation assumes the time-invariance hypothesis, which means the probability distribution at each moment can be parameterized using the same set of parameters [3].

[3]Due to the window length k, we actually need the states $S_{0:T+k-1}$ in order to fully reconstruct $U_{0:T-1}$. If the trajectory length of some agents is less than $T+k-1$, we pad their action sequences so they can participate in the training together.

predicted agent, and traffic light states, all information at each time is encoded into one token. Notably, we unified the spatial information of agents and map lines into a common spatial representation (CSR) by using consistent vectorized features. Agents are represented by vectors connecting consecutive corners of their bounding boxes, while map features are represented by vectors connecting consecutive line points. Thus, each object, including agents and map features, is represented as a collection of connected vectors, expressed as $\mathbf{V} = \{\mathbf{v}_i \mid \mathbf{v}_i = (p_i \rightarrow p_{i+1}), i = 1, 2, \ldots, m\}$, where p_i are the defining points of the object's geometry (e.g., bounding box corners or line points), and m is the number of vectors representing the object.

For encoding, we utilize a common subgraph encoder [9], termed the common spatial encoder, to simultaneously encode the spatial features of all objects in the scene. Attribute information, such as category labels and traffic light status, is encoded using a Multi-Layer Perceptron (MLP). These attribute features are then combined with the spatial features to produce a comprehensive token vector for each scene element.

Additionally, we embed the predicted agent's previous action u_{t-1} into its corresponding token, which we refer to as U-Embedding. Subsequently, we employ a multi-head self-attention mechanism [25] as the scene fusion module to facilitate interactions between various scene elements. These operations enhance the model's understanding of the environment and the agent's states. See Section IV-F for empirical evidence of their effectiveness.

Control action decoder Our model leverages an autoregressive temporal interaction module that utilizes multihead self-attention to learn complex relationships between scene tokens over the time series. It then decodes control action sequences using an MLP detokenizer. During training, to prevent information leakage, we employ causal attention, ensuring that each element can only interact with elements from previous time steps. Additionally, we apply teacher forcing [28], which inputs states from driving logs instead of predicted states to maintain consistency across timesteps.

Reactive inference with world state update To ensure the accurate capture of interactions among all agents, we update the world state of each agent-centric perspective before predicting future control actions. Formally, we infer the world state at time $t+1$, denoted as $\mathcal{S}^{\mathrm{w}}_{t+1}$, based on the states of N agents $\{S^{\mathrm{a}_i}_t\}^N_{i=1}$ and their control actions u at time t, in addition to the world state at time t, $\mathcal{S}^{\mathrm{w}}_t$, as described by the following equation:

$$\mathcal{S}^{\mathrm{w}}_{t+1} = \phi(\mathcal{S}^{\mathrm{map}}, \{\mathcal{K}(\mathcal{S}^{a_i}_t, u^{a_i}_t)\}^N_{i=1}) \tag{8}$$

Here, the control actions $\{u^{a_i}_t\}^N_{i=1}$ for all N agents are sampled from $P_\theta\left(U^{\mathrm{a}_i}_t \mid \mathcal{S}^{\mathrm{w}_i}_{\leq t}, \mathcal{S}^{\mathrm{a}_i}_{\leq t}\right)$, parameterized by our model[4]. This control action is then combined with the current state $\mathcal{S}^{\mathrm{a}}_t$ and the state transition law $\mathcal{K}$ to infer $\mathcal{S}^{\mathrm{a}}_{t+1}$. This process ensures the consistency of the world state for each agent at time $t+1$.

After inferring all agents' next states, we transform the world representation defined in Eq. (1) into each agent's local, agent-centric coordinate frame at time $t+1$, as defined by ϕ in Eq. (8). Modeling for each agent allows us to easily fine-tune the model to create drivers with distinct driving habits, which is extremely beneficial for downstream simulation and planning tasks. We demonstrate this flexibility in Section IV-E.

[4]All agents in the dataset, including vehicles, cyclists, pedestrians, and other obstacles of unknown categories, are treated as predicted agents. A unified set of parameters θ is used to estimate the action distribution of each agent.

E. Driving habit customization

KiGRAS can be further fine-tuned in post-training for better adaptation to practical applications, effectively functioning as a pre-trained model. Various techniques [18, 19] can be employed to fine-tune this model to accommodate different driving behaviors. Examples of these behaviors include avoiding collisions and driving quickly to meet the requirements of different scenarios. Here, we use the Discriminative Policy Optimization method [18].

Specifically, we leverage our pre-trained model to roll out the future trajectories of various prediction targets based on the scene state x_i (including map information, agent initialization states, etc.). We then apply expert rules to select winner and losser sample pairs (y^i_w, y^i_l), followed by preference fine-tuning using the loss function described in Eq. (9):

$$\begin{aligned}\mathcal{L}_{\mathrm{DPO}}\left(\pi_\theta; \pi_{\mathrm{ref}}\right) = -\,\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Bigg[\log\sigma\Bigg(\beta\log\frac{\pi_\theta\left(y_w \mid x\right)}{\pi_{\mathrm{ref}}\left(y_w \mid x\right)} \\ -\beta\log\frac{\pi_\theta\left(y_l \mid x\right)}{\pi_{\mathrm{ref}}\left(y_l \mid x\right)}\Bigg)\Bigg]\end{aligned} \tag{9}$$

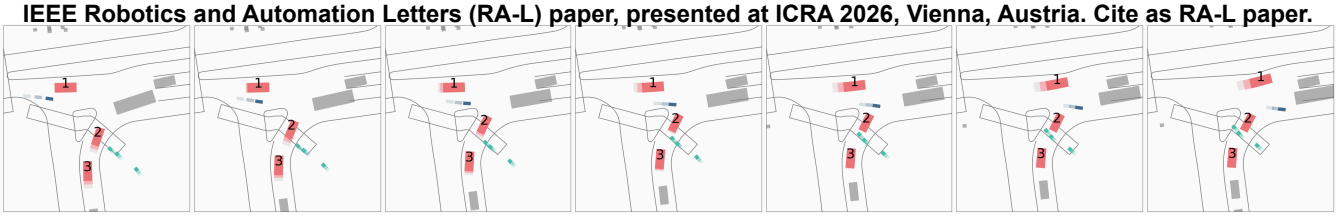
The model π_θ represents the policy we aim to optimize, while π_{ref} serves as the reference policy. In our approach, π_{ref} refer to our pre-trained model. The scaling factor β and the sigmoid function σ are employed to fine-tune preferences, encouraging the learned policy π_θ to favor trajectories y_w over y_l based on the reference policy π_{ref}.

IV. Experiment

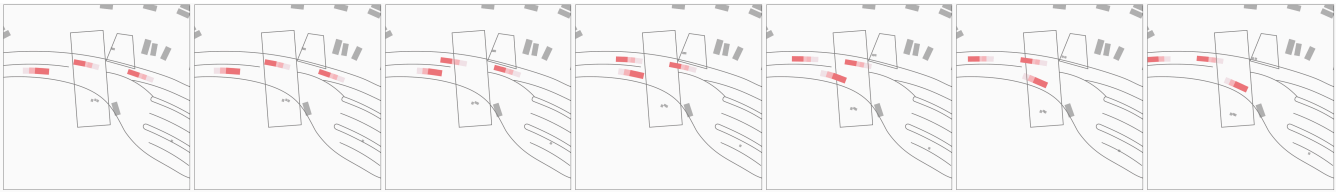
This section presents the experiment results and analysis. First, we introduce the dataset used and the metrics for subsequent quantitative analysis. Next, we perform a quantitative performance analysis of our method compared to several other modeling approaches. Then, we present closed-loop simulation results of our model in several classic scenarios for qualitative analysis. Following this, we demonstrate the fine-tuning results of specific driving styles based on the pre-trained driver model. Finally, we conduct ablation studies on various modules used during model training.

A. Dataset and metrics

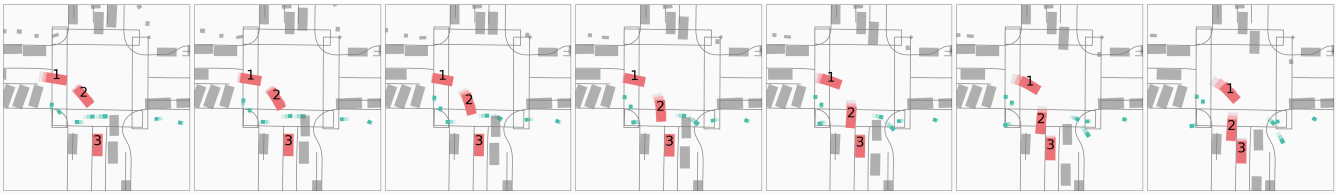
We conducted experiments on the Waymo Motion Dataset [23] v1.2, containing 103,354 real human driving scenarios. Our model was trained exclusively on the training set with a 2 Hz frequency. For fair comparison with other methods, we adopted the metrics provided by Waymo SimAgents to evaluate the trajectories of all agents over an 8-second closed-loop simulation. These metrics cover kinematic, agent interaction, and map-related aspects. The kinematic metric (Kin.) reflects how closely the simulated trajectories match real-world behavior in terms of linear and angular velocities and accelerations. The interaction metric (Inter.) assesses the quality of agent interactions, considering time-to-collision, distance to obstacles, and collision events. The map metric (Map.) reflects how true



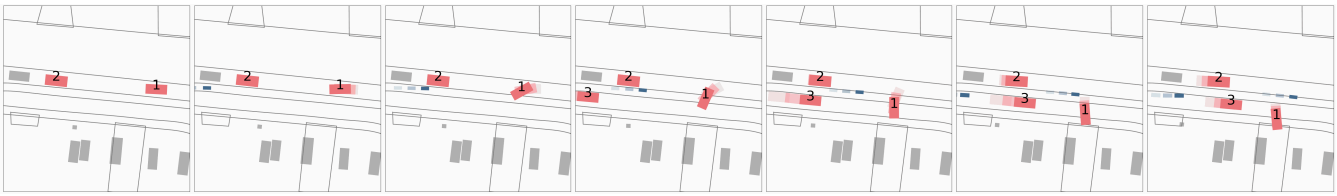
(a) Multi-agent interaction.



(b) Keep lane.



(c) Unprotected right turn.



(d) Car (index 1) driving against the flow of traffic.

Fig. 4. Qualitative results of closed-loop simulation. We present four representative scenarios generated by KiGRAS. Trajectories of all agents in these scenarios are generated by KiGRAS. In a fully closed-loop setting, we simulate the future for 8 seconds. Agents of interest are highlighted with distinct colors (some with labels), and their one-second historical trajectories are shown to illustrate their speed changes.

to reality the vehicle follows the road by measuring its distance to the road and whether it remains within roadway boundaries. REALISM (Real.) is an overall metric derived from a weighted combination of the aforementioned metrics, as explicitly stated by the Waymo SimAgents challenge. Other metrics include acceleration (Acc), jerk, and collision rate.

B. Implementation details

Each agent and map line are encoded into 64-dimensional vectors. For each prediction target, we focus on the 64 nearest obstacles. A 3-layer self-attention mechanism integrates all scene elements into a 64-dimensional scene token, followed by three layers of causal interaction.

During pre-training, the batch size is 256, and the learning rate is initialized at 2e-4, with the OneCycleLR scheduler dynamically adjusting it. Action probability space learning is supervised using cross-entropy loss. For inference, we aggregate all agent data along the batch dimension to facilitate parallel computation on the GPU. To reduce the complexity of causal attention, we utilize the key-value (KV) cache mechanism [17]. After optimization, we tested on an NVIDIA 3080 GPU with up to 64 agents and 100 map lines, achieving an inference time of 6.34 ms per step.

For DPO fine-tuning, the parameter β in Eq. (9) is set to 1. The batch size is 16, and the learning rate is 1e-6. Details on the selection of positive and negative samples are provided in Section IV-E.

C. Performance comparison

Due to changes in Waymo's metrics in 2024 and the closure of submissions to the 2023 leaderboard, we only compared our method with those appearing on the 2024 leaderboard for fairness, including SMART [29], BehaviorGPT [33], GUMP [12], MVTE [26], VBD [13] and TrafficBOTv1.5 [32]. The results are reported in Table I. Our model achieved advanced performance on the overall metric of the Waymo SimAgents challenge, surpassing contemporary methods. Additionally, our model has only 0.7M parameters, making it significantly smaller and more lightweight than both 2024 and previous models. This highlights the high parameter efficiency of our model.

D. Qualitative results

In this experiment, we present the results of closed-loop simulations over 8 seconds for all agents using the pre-trained driver model in four classic scenarios, as shown in Fig. 4. In Fig. 4a, the red vehicle 2 yields to the blue non-motorized vehicle, and the red vehicle 3 decelerates to allow pedestrians to cross. In Fig. 4b, three red vehicles maintain their lanes on a complex-shaped road. In Fig. 4c, red vehicles 1 and 2 perform unprotected right turns and yield to pedestrians, with vehicle 2

TABLE I
COMPARISON RESULTS OF KIGRAS AND STATE-OF-THE-ART APPROACHES IN SIMAGENTS CHALLENGE.

Method	Real.	Kin.	Inter.	Map.	minADE
SMART-96M	0.7564	0.4769	0.7986	0.8618	1.5501
SMART-7M	0.7591	0.4759	0.8039	0.8632	**1.4062**
BehaviorGPT-3M	0.7473	0.4333	0.7997	0.8593	1.4147
GUMP-523M	0.7431	**0.4780**	0.7887	0.8359	1.6041
MVTE	0.7302	0.4503	0.7706	0.8381	1.6770
VBD-12M	0.7200	0.4169	0.7819	0.8137	1.4743
TrafficBOTv1.5-10M	0.6988	0.4304	0.7114	0.8360	1.8825
KiGRAS-0.7M	**0.7597**	0.4691	**0.8064**	**0.8658**	1.4383

attempting an efficient lane change to overtake vehicle 3. In this case, we also observed an issue where pedestrians tend to have close encounters with the edges of motor vehicles.

Additionally, we discovered a particularly interesting case in the Waymo test set where vehicle 1 stopped against the flow of traffic, a scenario rarely seen in the training set (shown as Fig. 4d). Remarkably, our model demonstrated a degree of generalization to this corner case; vehicle 1 promptly made a left turn into a parking space on the left side, allowing other vehicles to proceed smoothly.

E. Results of driving habit customization

We fine-tuned our model using the DPO method introduced in Section III-E. Based on KiGRAS trained on the Waymo Motion training dataset, denoted as the pre-trained driver (PDriver) model, we developed three specialized drivers: the safety driver (SDriver), optimized for collision avoidance; the fast driver (FDriver), optimized for maximizing speed; and the comfort driver (CDriver), optimized for minimizing jerk.

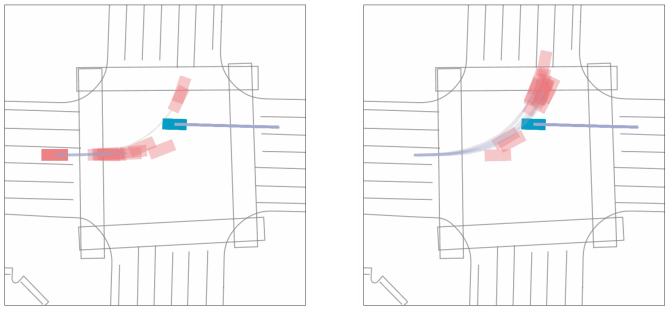
Initially, we sampled 256 potential future trajectories from the pre-trained driver model using a top-p sampling strategy, with data from other agents sourced from log data. For SDriver, we selected a non-collision trajectory as the positive sample and a collision trajectory as the negative sample. The approach for selecting positive and negative samples for FDriver was slightly different: we selected the fastest non-collision trajectory as the positive sample, and a trajectory with a lower speed than the positive sample as the negative sample. For CDriver, we chose the trajectory with the lowest maximum jerk that did not result in a collision as the positive sample, and a trajectory with a higher maximum jerk than the positive sample as the negative sample. We conducted closed-loop simulations to evaluate the performance of each specialized driver on the Waymo test dataset. Specifically, we used each custom driver to control the ego-vehicle, while all other agents were controlled by PDriver to ensure a fair comparison. We selected the action with the maximum probability from the model's predicted action space. Results are shown in Table II.

We observed that safety driver demonstrated a 1.6052‰ reduction in collision rate over 8 seconds compared to PDriver. Conversely, fast driver achieved a 0.4872 m/s increase in average speed; however, this speed increase resulted in a higher collision rate. Additionally, comfort driver exhibited a 21.5% decrease in average jerk and a 20.5% decrease in maximum jerk.

To provide a clearer understanding of the changes before and after fine-tuning, we conducted 16 times of semi-closed-loop simulations over 8 seconds. In these simulations, the red ego vehicle was controlled by the model, while the blue other agents were replayed from logs. The results are illustrated in Fig. 5, where it is evident that fast driver operates at a significantly higher speed.

TABLE II
COMPARISON WITH DIFFERENT STYLE DRIVER MODELS.

Model	Speed (m/s)	Acc (m/s^2)	Jerk (m/s^3)	$Jerk_{max}$ (m/s^3)	Collision Rate (‰) 3s	5s	8s
PDriver	5.211	0.402	0.065	0.195	2.118	4.169	7.959
SDriver	5.534	0.413	0.077	0.231	1.850	3.389	6.354
CDriver	5.017	0.364	0.051	0.155	2.096	4.236	8.271
FDriver	5.707	0.440	0.076	0.223	2.185	4.860	9.676



(a) Pre-trained driver. **(b)** Fast driver.

Fig. 5. Performance Comparison of pre-trained driver and fast driver models. Both models were simulated 16 times for 8 seconds each under semi-closed-loop settings.

F. Ablation study

We conducted an ablation study on three key components of our pre-trained model using the validation dataset. The results are presented in Table III. Notably, the Causal-Attn component led to a significant improvement in the overall metric Real., increasing from 0.7419 to 0.7548, along with consistent gains across other sub-metrics. Incorporating CSR feature representations for agents and the map further increased the Real metric to 0.7555, with gains observed in other sub-metrics. Adding the U-Embedding component resulted in a slight improvement in the Real score from 0.7555 to 0.7558. Additionally, we observed a modest increase in the Inter. score (from 0.7988 to 0.7995), while the Kin. and Map. scores remained largely unchanged. The U-Embedding may help the model capture temporal patterns in control actions, potentially reducing control violations, which could explain the slight decrease in the minADE metric. In the Waymo dataset, most agents are stationary, and moving agents typically follow linear motion with minimal control variation. In such cases, Causal-Attn may stabilize action generation. Data imbalance and the evaluation method which assesses all agents likely limit improvements in CSR and U-Embedding.

TABLE III
ABLATION FOR VARIOUS MODULES IN PRE-TRAINED DRIVER MODEL.

ID	Causal-Attn	CSR	U-Embedding	Real.	Kin.	Inter.	Map.	minADE
1				0.7419	0.4581	0.7823	0.8521	1.9505
2	✓			0.7548	0.4684	0.7980	0.8629	1.5181
3	✓	✓		0.7555	0.4697	0.7988	**0.8631**	**1.4909**
4	✓	✓	✓	**0.7558**	**0.4698**	**0.7995**	**0.8631**	1.5041

V. Conclusions and Future work

This paper proposes KiGRAS, a novel autoregressive trajectory generation paradigm that transforms the task to model the distribution of control actions, thereby capturing the physical causality of trajectories. This new task definition offers a much more compact representation of realistic driving patterns, fundamentally reducing task complexity. Additionally, KiGRAS inherently ensures the physical feasibility of generated trajectories through task re-formulation, providing a significant advantage over previous methods. KiGRAS achieves top performance in the Waymo SimAgents Challenge with a significantly smaller parameter scale, opening a new frontier for the development of next-generation Deep Learning (DL) paradigms in the autonomous driving domain.

For future work and applications, our model has the potential to be used as a simulator trained on real driving data, which opens up opportunities for its application in reinforcement learning tasks. Additionally, our approach can be easily extended to motion prediction and planning tasks. A key focus of our research will be on how to sample actions from the action space to meet specific task requirements effectively. Additionally, we will explore methods to accelerate the model, including scene-centric representations [5] to enhance computational efficiency. Furthermore, we will investigate strategies to bridge the gap between kinematic models and real-world vehicles.

References

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat, *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[3] G. Benton, M. Finzi, P. Izmailov, and A. G. Wilson, "Learning invariances in neural networks from training data," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 17 605–17 616, 2020.

[4] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Adv. Neural Inf. Process. Syst.*, vol. 33, pp. 1877–1901, 2020.

[5] Y. Chen, S. Tonkens, and M. Pavone, "Categorical traffic transformer: Interpretable and diverse behavior prediction with tokenized latent," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*. IEEE, 2025, pp. 2423–2430.

[6] R. Cimurs, I. H. Suh, and J. H. Lee, "Goal-driven autonomous exploration through deep reinforcement learning," *IEEE Robot. Autom. Lett.*, vol. 7, no. 2, pp. 730–737, 2021.

[7] B. D. Evans, H. A. Engelbrecht, and H. W. Jordaan, "High-speed autonomous racing using trajectory-aided deep reinforcement learning," *IEEE Robot. Autom. Lett.*, 2023.

[8] C. Feng, H. Zhou, H. Lin, Z. Zhang, Z. Xu, C. Zhang, B. Zhou, and S. Shen, "MacFormer: Map-agent coupled transformer for real-time and robust trajectory prediction," *IEEE Robot. Autom. Lett.*, 2023.

[9] 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 *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, June 2020.

[10] C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—a survey," *Automatica*, vol. 25, no. 3, pp. 335–348, 1989.

[11] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *J. Field Robot.*, vol. 37, no. 3, pp. 362–386, 2020.

[12] Y. Hu, S. Chai, Z. Yang, J. Qian, K. Li, W. Shao, H. Zhang, W. Xu, and Q. Liu, "Solving motion planning tasks with a scalable generative model," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*. Springer, 2024, pp. 386–404.

[13] Z. Huang, Z. Zhang, A. Vaidya, Y. Chen, C. Lv, and J. F. Fisac, "Versatile scene-consistent traffic scenario generation as optimization with diffusion," *arXiv preprint arXiv:2404.02524*, 2024.

[14] Y. Liu, X. Pei, H. Zhou, and X. Guo, "Spatiotemporal trajectory planning for autonomous vehicle based on reachable set and iterative lqr," *IEEE Trans. Veh. Technol.*, 2024.

[15] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Trans. Intell. Veh.*, vol. 1, no. 1, pp. 33–55, 2016.

[16] J. Philion, X. B. Peng, and S. Fidler, "Trajeglish: Learning the language of driving scenarios," *arXiv preprint arXiv:2312.04535*, 2023.

[17] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[18] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," *Adv. Neural Inf. Process. Syst.*, vol. 36, 2024.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[20] A. Seff, B. Cera, D. Chen, M. Ng, A. Zhou, N. Nayakanti, K. S. Refaat, R. Al-Rfou, and B. Sapp, "MotionLM: Multi-agent motion forecasting as language modeling," in *Proc. Int. Conf. Comput. Vis. (ICCV)*, 2023, pp. 8579–8590.

[21] H. W. Stone, *Kinematic modeling, identification, and control of robotic manipulators*. Springer Science & Business Media, 2012, vol. 29.

[22] J. Sun, C. Yuan, S. Sun, S. Wang, Y. Han, S. Ma, Z. Huang, A. Wong, K. P. Tee, and M. H. Ang, "ControlMTR: Control-guided motion transformer with scene-compliant intention points for feasible motion prediction," in *Proc. IEEE 27th Int. Conf. Intelligent Transp. Syst. (ITSC 2024)*. IEEE, 2024, pp. 1507–1514.

[23] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2446–2454.

[24] C. Tang, D. Huang, W. Ge, W. Liu, and H. Zhang, "Graspgpt: Leveraging semantic knowledge from a large language model for task-oriented grasping," *IEEE Robot. Autom. Lett.*, 2023.

[25] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[26] Y. Wang, T. Zhao, and F. Yi, "Multiverse Transformer: 1st place solution for waymo open sim agents challenge 2023," *arXiv preprint arXiv:2306.11868*, 2023.

[27] X. Weng, Y. Yuan, and K. Kitani, "PTP: Parallelized tracking and prediction with graph neural networks and diversity sampling," *IEEE Robot. Autom. Lett.*, vol. 6, no. 3, pp. 4640–4647, 2021.

[28] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[29] W. Wu, X. Feng, Z. Gao, and Y. Kan, "SMART: Scalable multi-agent real-time motion generation via next-token prediction," *Adv. Neural Inf. Process. Syst.*, vol. 37, pp. 114 048–114 071, 2024.

[30] P. Xu, J.-B. Hayet, and I. Karamouzas, "Context-aware timewise vaes for real-time vehicle trajectory prediction," *IEEE Robot. Autom. Lett.*, 2023.

[31] R. Yang, J. Wang, Z. Geng, M. Ye, S. Ji, B. Li, and F. Wu, "Learning task-relevant representations for generalization via characteristic functions of reward sequence distributions," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discov. Data Min.*, 2022, pp. 2242–2252.

[32] Z. Zhang, C. Sakaridis, and L. Van Gool, "TrafficBots V1.5: Traffic simulation via conditional vaes and transformers with relative pose encoding," *arXiv preprint arXiv:2406.10898*, 2024.

[33] Z. Zhou, H. Haibo, X. Chen, J. Wang, N. Guan, K. Wu, Y.-H. Li, Y.-K. Huang, and C. J. Xue, "BehaviorGPT: Smart agent simulation for autonomous driving with next-patch prediction," *Adv. Neural Inf. Process. Syst.*, vol. 37, pp. 79 597–79 617, 2024.

[34] Z. Zhou, Z. Wen, J. Wang, Y.-H. Li, and Y.-K. Huang, "QCNeXt: A next-generation framework for joint multi-agent trajectory prediction," *arXiv preprint arXiv:2306.10508*, 2023.

[35] R. Zhu, J. Zhao, D. Zhang, G. Wang, X. Chen, S. Zhang, J. Gong, Q. Zhou, W. Zhang, N. Wang, *et al.*, "SparseAD: Sparse query-centric paradigm for efficient end-to-end autonomous driving," *IEEE Trans. Artif. Intell.*, 2025.

[36] J. Zhuang, G. Wang, S. Zhang, X. Wang, H. Zhou, Z. Xu, C. Zhang, and Z. Li, "StreamMOTP: Streaming and unified framework for joint 3d multi-object tracking and trajectory prediction," in *Proc. Asian Conf. Comput. Vis. (ACCV)*, 2024, pp. 3189–3205.