

# ChatDyn: Language-Driven Multi-Actor Dynamics Generation in Street Scenes

Yuxi Wei<sup>1</sup> Jingbo Wang<sup>2†</sup> Yuwen Du<sup>1</sup> Dingju Wang<sup>1</sup>  
 Liang Pan<sup>2</sup> Chenxin Xu<sup>1</sup> Yao Feng<sup>3,4</sup> Bo Dai<sup>2</sup> Siheng Chen<sup>1†</sup>

<sup>1</sup> Shanghai Jiao Tong University <sup>2</sup> Shanghai AI Laboratory

<sup>3</sup> Max Planck Institute for Intelligent Systems <sup>4</sup> ETH Zürich

{wyx3590236732, wangdingju, xcwkak, sihengc}@sjtu.edu.cn, duyuwen@tju.edu.cn

{wangjingbo, panliang, daibo}@pjlab.org.cn, yao.feng@tuebingen.mpg.de

[vfishc.github.io/chatdyn](https://vfishc.github.io/chatdyn)

## Abstract

Generating realistic and interactive dynamics of traffic participants according to specific instruction is critical for street scene simulation. However, there is currently a lack of a comprehensive method that generates realistic dynamics of different types of participants including vehicles and pedestrians, with different kinds of interactions between them. In this paper, we introduce ChatDyn, the first system capable of generating interactive, controllable and realistic participant dynamics in street scenes based on language instructions. To achieve precise control through complex language, ChatDyn employs a multi-LLM-agent role-playing approach, which utilizes natural language inputs to plan the trajectories and behaviors for different traffic participants. To generate realistic fine-grained dynamics based on the planning, ChatDyn designs two novel executors: the PedExecutor, a unified multi-task executor that generates realistic pedestrian dynamics under different task planings; and the VehExecutor, a physical transition-based policy that generates physically plausible vehicle dynamics. Extensive experiments show that ChatDyn can generate realistic driving scene dynamics with multiple vehicles and pedestrians, and significantly outperforms previous methods on subtasks. Code and model will be available at <https://vfishc.github.io/chatdyn>.

## 1. Introduction

Autonomous driving simulation plays a critical role in the training and validation of driving systems, which attracts significant attention in recent years. To create a realistic simulation system, it must replicate real world scenarios, in-

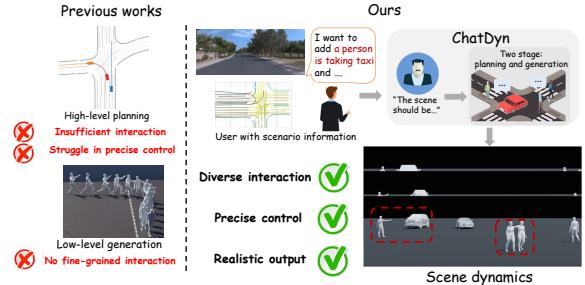


Figure 1. ChatDyn achieves interactive and realistic language-driven multi-actor dynamics generation in street scenes.

cluding the dynamics of various traffic participants such as pedestrians and vehicles [21, 23, 24], as well as their interactions. These dynamics are essential for constructing realistic scene events, maintaining the temporal consistency of simulations [8, 14, 60, 73], and enabling accurate decision making [2, 48, 66] in autonomous systems. However, generating such dynamics is challenging, as it involves modeling the behavior of vehicles and pedestrians individually while also capturing their intricate interactions within street scenes. This requires high-level planning to coordinate trajectory generation for both types of participants, taking the street scenes into account, and precise low-level control to ensure that their interactions appear realistic.

Existing methods often focus on one of these aspects while neglecting the other. High-level planning approaches, such as LCTGen [63] and CTG++[82], use language-based inputs to generate scene-level plans. However, they struggle to control detailed participant behaviors and often lack diversity in participants, such as pedestrians, as well as the varied interactions between different participants. On the other hand, low-level generation methods, such as Pacer[56] and Pacer+ [69], excel at producing fine-grained pedestrian dynamics based on specific control signals. Yet, they fail

<sup>†</sup>Corresponding authors.

to account for the nuanced interactive behaviors that arise between pedestrians and other participants.

In this work, we introduce ChatDyn, the first system achieving interactive and realistic language-driven multi-actor dynamics generation in street scenes, as shown in Fig. 1. The key insight behind ChatDyn is the need to combine both high-level planning and low-level control to create a realistic simulation system. Large language models (LLMs) excel at high-level planning, while physics-based methods are well-suited for fine-grained, low-level control. Building on this observation, ChatDyn combines a multi-LLM agent for high-level planning with physics-based executors for low-level generation. By integrating high- and low-level components, ChatDyn allows users to input language instructions to specify their requirements. It then generates realistic dynamics for diverse traffic participants, including pedestrians and vehicles, within the specified scene, while accommodating complex and varied interactions among participants.

Specifically, ChatDyn employs multi-LLM-agent role-playing for high-level planning to understand complex and abstract user instructions. The key idea is to treat each traffic participant as an agent, which can extract specific requirements and semantics from the language instructions, interact according to the requirement of scenes, and generate planned trajectories and behaviors with interactive characteristics. The multi-LLM-agent role-playing approach shows two notable advantages: first, LLM has powerful language understanding capabilities, enabling them to accurately capture the complex and abstract demands in user instructions, thus producing planning results that closely align with them; second, the interaction between agents allows for the incorporation of interactive information during the planning process, ensuring that the planning results, at the high-level stage, exhibit interactive characteristics.

Due to the different characteristics of the dynamics of pedestrians and vehicles, we design separate executors for their low-level generation. To generate realistic and interactive fine-grained low-level pedestrian dynamics, we propose PedExecutor, a unified method accomplishes multiple high-level planned tasks, including fine-grained physical interactions, and returns human-like dynamics. PedExecutor is a physics-based control policy that fulfills multiple control requirements and enables interactive tasks for pedestrians under realistic physical feedback. A task masking mechanism and related training strategy make it handle different requirements with a single unified policy. PedExecutor also incorporates hierarchical control and body-masked adversarial motion prior, which introduce priors to both the action space and reward space, allowing for the generation of realistic, natural dynamics while executing the planning.

To obtain physically feasible low-level vehicle dynamics, we propose VehExecutor, a control policy based on

vehicle physical transition process. VehExecutor generates vehicle actions from high-level trajectory planning, and physical transition process returns the next vehicle state which can be accumulated as final dynamics output. VehExecutor ensures the physical feasibility of output dynamics and eliminating unrealistic factors such as tail swings and unwanted translations. Additionally, VehExecutor incorporates history-aware state and action design, simulating the real-world decision-making process of vehicles, which improves the accuracy and consistency of control.

Based on the above designs, ChatDyn satisfied the three required characteristics: for interactivity, multi-LLM-agent role-playing enables interaction planning at the high-level, while the executors execute the interaction-aware plans to generate low-level interactive behaviors; for controllability, LLM-agents precisely follow the user’s natural language inputs, and the executors precisely execute the planning for accurate control; for realism, PedExecutor employs hierarchical control and integrates body-masked adversarial motion priors, producing dynamics that align with human intuitions, while VehExecutor highly simulates the physical feedback and kinematic characteristics of real vehicle.

Our contributions can be summarized as: (1) We design the first language-driven dynamics generation system for multi-actor in street scenes, utilizing multi-LLM-agent role-playing for high-level planning. (2) We propose a unified pedestrian executor that, under hierarchical control, executes multiple planning tasks and generates realistic low-level pedestrian dynamics based on planning. (3) We propose a vehicle executor that derives realistic control processes and low-level vehicle dynamics from planning trajectories. (4) We demonstrate the realistic language-driven outputs of the system in experiments and validated the effectiveness of its each individual component.

## 2. Related Works

**Human dynamics generation.** The generation of human dynamics can broadly be divided into kinematics-based and physics-based approaches. For kinematics-based generation, transformer-based methods [3, 25, 26] and diffusion models [11, 33, 80, 81] are used to generate corresponding dynamics based on language inputs. Recent methods [58, 68, 74] further introduce more control conditions, while others [12, 85] focus on improving efficiency in the generation process. However, these approaches do not account for physical constraints, often resulting in physically implausible results, particularly for interactions. For physics-based generation, existing work like [15, 44, 45, 51, 52, 71, 79] achieve predefined tasks with plausible dynamics. [4, 32, 65, 75, 76] extend the human character dynamics content with different designs. [56, 69] focus on the dynamics of street scene pedestrians. However, these methods do not involve the interaction behaviors

between multi-pedestrians. Our PedExecutor considers the interaction behaviors, and is trained as a unified policy for multiple scenarios also including following, imitation.

**Vehicle dynamics generation.** In industry, vehicle dynamics are usually generated by software tools [9, 19, 31, 54]. Some research works [6, 17, 55, 62] focus on unconditioned generation to simplify the generation process. Recent works on vehicle dynamics generation under different conditions have introduced various approaches. Traffic-Sim [61] uses predictive models to generate vehicle dynamics in a scene, while CTG [83] introduces diffusion models into the generation process. CTG++ [82] and LCTGen [63] achieve scene-vehicle dynamics generation controlled by language, and SceneControl [43] adds multiple controllable parameters to the generation process. RealGen [13] uses a retrieval-based approach to improve the realism of generated dynamics. However, these works generally do not directly consider physical constraints and kinematic constraints, and lack sufficient interaction. And these language-controlled methods cannot achieve precise control over different vehicles. Our LLM-agents planning utilizes the language understanding capability of LLMs to achieve precise control of participants considering interactive information, and VehExecutor generates the final physical plausible vehicle dynamics.

**Large language models and agents.** Recently, numerous large language models [1, 5, 22, 39, 67, 78] are proposed and released. Many works have leveraged these models by finetuning them [18, 30, 40, 53] or integrating them with relevant tools to build LLM-based agents [41, 72] capable of performing targeted tasks or specific functions. These agents have been applied to a wide range of downstream tasks, including code development [28, 29], web browsing [34, 84], medical Q&A [36, 37], social behavior simulation [35, 50], and issue handing [59, 64]. In this paper, we explore the application of agents in traffic simulation, utilizing them to interact and perform trajectory and behavior planning within traffic scenarios.

### 3. Method

#### 3.1. System Overview

ChatDyn interprets and analyzes user language instructions, then produces scene dynamics that align with them. ChatDyn employs a two-stage process: high-level planning which plans trajectory and behavior under complex and abstract command; low-level generation for fine-grained, realistic dynamics generation. Since user instructions may contain many specific details that require precise control and abstract semantics that need to be understood, ChatDyn leverages multi-LLM-agent role-playing, treating each traffic participant as an LLM-agent. This approach capitalizes on the LLM’s ability to comprehend semantic informa-

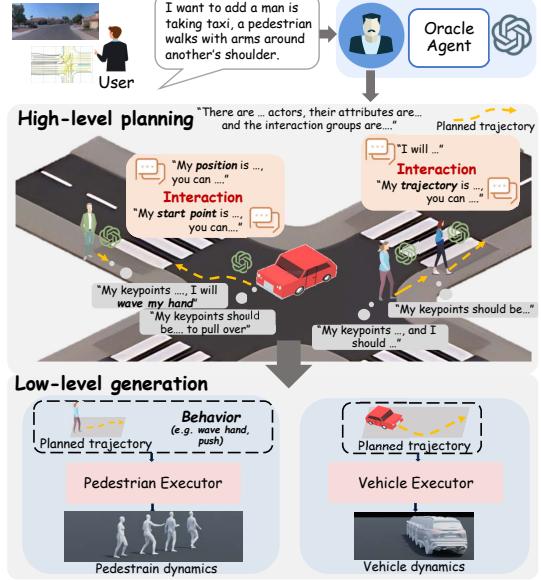


Figure 2. System overview. ChatDyn adopts multi-LLM-agent role-playing for precise high-level planning. Two specialized executors are designed for realistic low-level generation.

tion and its extensive commonsense priors, using specific tools and interaction process to complete high-level trajectory and behavior planning. Each traffic participant’s corresponding agent is also equipped with an executor as one of the tools. After the high-level planning is completed, the executor uses the planning results to execute the low-level generation process. The executors generate fine-grained, realistic, and physically feasible dynamics based on high-level planning. See Fig. 2 for illustration.

#### 3.2. High-Level Planning

High-level planning is completed by LLM-agents and interaction among them. Each LLM-agent consists of an LLM component and the relevant tools, and the agents are divided into two types: oracle agents and actor agents. The oracle agent is responsible for directly receiving user instruction and understanding, dispatching them. Oracle agent breaks the instructions down into specific information for each participant, identifies interaction relationships, forms interaction groups, and generates the planning order schedule for the following process. Each actor agent corresponds to a specific traffic participant, initialized with the information provided by the oracle agent. The actor agents then interact with others based on the interaction groups to get information from others, and finally plan trajectory and behavior using their tools. The following parts will provide a detailed description of these two types of agents.

##### 3.2.1. Oracle Agent

Oracle agent breaks down complex and lengthy user instructions into specific actions for each participant, ensur-

ing an accurate understanding of the user’s instructions and generating the subsequent schedule. For its LLM component, we explain its responsibilities, outline our requirements and give some few-shot examples. The LLM component of the oracle agent outputs the initialization information for each participants, interaction groups, and the schedule in natural language. To facilitate subsequent execution, we equip it with a structured output tool that converts language output into a predefined data structure for execution. Oracle agent provides the system with the ability to handle complex, mixed, and abstract instructions, and streamlines operations for clarity and fine granularity. Detailed instruction prompt can be found in the appendix.

### 3.2.2. Actor Agent

Each actor agent corresponds to a specific participant in the scene and is initialized with information provided by the oracle agent, including the agent type and description about its trajectory and behavior. The actor agents interact based on the oracle agent’s schedule to collect interactive information and completes the final high-level planning. The planning process of the actor agent incorporates map information, with the scene map represented as a graph  $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ , where  $\mathcal{N}$  represents different lane sections. Each node in  $\mathcal{N}$  contains the point set for the lane section and other extracted information such as its orientation and driving type (e.g., straight, left/right turn, lane/boundary). The edges  $\mathcal{E}$  represent the relationships between the lane sections, such as adjacency or connectivity. The specific planning process uses a keypoints-based approach, integrating actor interactions to complete the planning.

**Keypoints-based trajectory planning.** Intuitively, a trajectory is typically determined by a series of keypoints, which are then interpolated to form the complete trajectory. Specifically, the actor agent first obtains keypoints and then uses interpolation to get the planned trajectory. The LLM component of the actor agent infers the number of keypoints needed based on its initialization information, and then determines the generation process for each keypoint. There are generally two sources for keypoints: one is directly obtained from map and previous keypoints, where the actor agent retrieves keypoints from the map  $\mathcal{G}$  based on the keypoint’s attributes and relationship with the previous keypoint. The other source is through interaction, where a keypoint depends on information from others. This kind of keypoints are determined by actor interaction.

**Multi-actor interaction.** Interactions allow agents to acquire information from other agents to generate certain keypoints, thereby fulfilling the interaction requirements during the high-level planning process. The groups that need to interact have already been specified by the oracle agent. During the interaction, the LLM determines which pieces of information are needed from the interacting agents for a specific keypoint, such as the start point, endpoint, or

the entire trajectory. Each agent then shares the required information, and this exchanged information is provided to the corresponding tools that generate keypoints based on the agent’s own needs. Actor interaction enables interactive trajectory generation, which is common in the real scenarios.

After obtaining all desired keypoints, interpolation is applied to generate a complete trajectory, fulfilling the trajectory planning stage. Here, we employ Bézier curves for interpolation. Note that: (i) the interpolated trajectory at this stage lacks physical constraints and serves only as a preliminary plan to be executed by the subsequent executor; (ii) for static actors, two keypoints are required to determine orientation, as their facing direction must be specified; (iii) when the actor type is pedestrian, besides trajectory, additional behavioral instructions may be inferred by LLM component based on the initialization information. These behaviors could stem from direct command-based controls or indirectly from semantic cues, with the final instructions provided in text form for the executor to execute.

## 3.3. Low-Level Generation for Pedestrian

The pedestrian executor (PedExecutor), as shown in Fig. 3, generates low-level pedestrian dynamics based on the trajectories and behaviors planned from high-level planning. Pedestrian behaviors can be subdivided into single-agent behavior directly specified by language and interactive behaviors that occur between multiple agents. Thus the challenge lies in simultaneously handling trajectory following, single-agent motion specification, and multi-agent interactions, while maintaining human-like quality. To achieve these, PedExecutor utilizes multi-task unified training to execute the trajectory, single-agent behavior, and multi-agent interactions as planned by the LLM with a single policy. For human-like quality, the action space incorporates hierarchical control to provide priors, while the reward function uses body masked Adversarial Motion Priors (AMP) [51] to encourage human-like control. Ultimately PedExecutor returns realistic dynamics that follow planned trajectory and complete desired behaviors.

Control process is defined by Markov decision process  $\mathcal{M}^p = \{\mathcal{S}^p, \mathcal{A}^p, \mathcal{T}^p, \mathcal{R}^p, \gamma^p\}$ , where the elements represent states, actions, transition process, reward, and the discount factor. Goal-conditioned reinforcement learning (RL) is adopted for training. The transition process is implemented by the physics engine. Others are detailed below.

**States and multi-task unified training.** Our control policy is designed to handle multiple distinct tasks, requiring specialized processing and training to achieve a unified policy capable of addressing different tasks. The tasks are divided into three main categories based on requirements: trajectory following, single-agent behavior specification, and multi-agent interaction. For single-agent behavior, the LLM generates corresponding text descriptions,

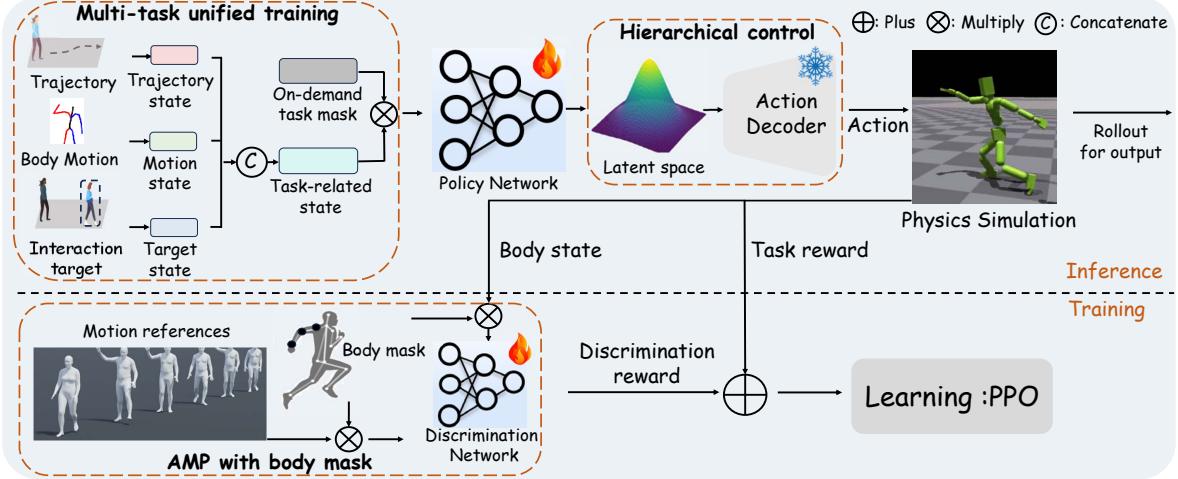


Figure 3. Pedestrian executor (PedExecutor) framework. With multi-task unified training, PedExecutor achieves unified control over various task, including following, imitation and interaction. It generates realistic pedestrian dynamics by effectively executing tasks derived from planning. Hierarchical control and AMP with body mask provide prior to action space and reward space, improving the realism of dynamics output.

which are then processed by a Text2Motion model (e.g., MoMask [26]) to produce an upper-body motion for the policy to imitate thereby achieve single-agent behavior specification. Multi-agent interaction tasks are trained according to predefined types, with the LLM’s planning classifying interaction behaviors into these types, enabling the executor to execute specific interactions as specified.

Task-related states consist of trajectory slices to be followed  $S_{traj}^p$ , motion to be imitated  $S_{mo}^p$ , and states of interacting targets  $S_{tar}^p$ .  $S_{traj}^p$  includes K future steps of planned trajectory from the current timestep, while  $S_{mo}^p = \text{concat}(\hat{j}_{pos}, \hat{j}_{rot}, \hat{j}_{vel}, \hat{j}_{\omega})$  comprises joint position  $\hat{j}_{pos}$ , joint rotation  $\hat{j}_{rot}$ , joint velocity  $\hat{j}_{vel}$ , and rotation velocity  $\hat{j}_{\omega}$  from the motion, with optional joint masking to retain only relevant parts, such as upper body.  $S_{tar}^p = \text{concat}(r_{pos}, r_{rot}, r_{vel}, r_{\omega}, r_{bbox}, e_{inter})$  includes the interacting target’s root position  $r_{pos}$ , root rotation  $r_{rot}$ , root velocity  $r_{vel}$ , root angular velocity  $r_{\omega}$ , and the bounding box’s 8 vertices  $r_{bbox}$ . Additionally, an interaction task embedding  $e_{inter}$  represented as a one-hot vector to specify the interaction type. Beyond task-related states, the final observations also include humanoid proprioception [69]  $S_{prop}^p$  to capture necessary internal states of the humanoid. The final  $S^p = \text{concat}(S_{traj}^p, S_{mo}^p, S_{tar}^p, S_{prop}^p)$

To activate specific parts during training/testing, we introduce a task masking mechanism. During training, tasks to be executed for each episode are sampled, and the remaining tasks related states are masked. Only the relevant environment for the involved tasks is prepared, and rewards are calculated based on that. During inference, states are activated as needed, while others are masked, enabling task-specific execution. This approach ensures the trained unified policy handles all of the tasks without confusion. After

sufficient training, its performance matches that of policies trained on individual tasks, while can also execute multiple non-conflicting tasks in a single run.

**Action hierarchical control.** Pedestrian action spaces are typically modeled using a proportional-derivative (PD) controller at each degree of freedom (DoF), but such spaces lack inherent priors, often leading to locally unrealistic actions for completing specific tasks. To address this, we apply a hierarchical action control space from PULSE [45]. The policy network first outputs to a pretrained latent space, obtaining a latent feature  $f_{action}$ , which is then decoded by a pretrained decoder into control signals. The pretrained latent space, equipped with the corresponding decoder, ensures that the decoded output distribution closely matches the input data from pretraining, thereby providing the action space with a real-world action prior.

**Reward design.** The reward consists of two main components. The first is the discrimination reward  $R_{disc}$  used to implement AMP [51], which employs a discriminator to encourage the policy to generate output that align with movement patterns observed in a dataset of human-recorded data clips. The second component is the task-related reward  $R_{task}$ , designed to motivate the policy to accomplish specific tasks for executing high-level planning. The detailed task reward designs can be found in appendix. Similar to [69], we implement early termination if excessive root distance error or joint distance error occurs during following or imitation.

**AMP with body mask and warm-up training.** We employ Proximal Policy Optimization (PPO) [57] for overall training optimization. However, directly optimizing for interaction tasks presents certain challenges: without using AMP, the lack of a discrimination reward makes it difficult

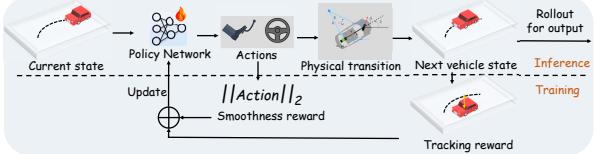


Figure 4. Vehicle executor (VehExecutor) framework. VehExecutor adopts goal-conditioned RL based on physical transition of real vehicle. Combining with history-aware design, VehExecutor generates realistic vehicle dynamics under planned trajectory.

to achieve human-like dynamics results. Conversely, proper reference data clip needed for specific tasks are usually complex and not easily obtainable. Using reference data clip from the following or imitation tasks can lead to misalignment with the actual requirements for interaction, resulting in a policy that fails to accomplish interaction tasks.

To address these issues, we adopt two training techniques: (i) Initially, we train without AMP. While this approach may not yield realistic dynamics, it allows the policy to complete interaction tasks, providing a more accurate initial domain for subsequent optimization. Once this foundation is established, we incorporate AMP training, which maintains task completion while improving dynamics quality. (ii) We introduce AMP body mask, where certain joints are excluded during the AMP calculations. Since interaction tasks primarily engage specific body part(e.g. arm), we apply the AMP mask to the relevant body parts, ensuring that the rest parts of dynamics can leverage AMP for more natural movements while allowing the specific part to complete the tasks. By utilizing these two training techniques, we achieve a policy that satisfies both visual realism and the ability to perform interaction tasks effectively.

### 3.4. Low-Level Generation for Vehicles

The vehicle executor (VehExecutor), as shown in Fig. 4, generates the final realistic and physically feasible vehicle dynamics with control policy based on the high-level planned trajectory, which may initially violate certain dynamic constraints. To involve physical constraints and achieve precise control, VehExecutor utilizes physics-based transition environment, combined with history-aware state and action space design. The final dynamics can be obtained by accumulating the vehicle position and heading from environment. The control process also modeled as a Markov decision process defined by  $\mathcal{M}^v = \{\mathcal{S}^v, \mathcal{A}^v, \mathcal{T}^v, \mathcal{R}^v, \gamma^v\}$ , with goal-conditioned RL for training. Further details are provided below.

**History-aware states.** The VehExecutor incorporates historical information into its states, alongside its current state, to improve the temporal consistency of the policy. The vehicle states consist of the planned trajectory segment  $\hat{\mathbf{P}}^v$ , temporal velocity  $\mathbf{V}^v$ , and dynamic parameters  $\Theta^v$ .  $\mathbf{P}^v$  is a slice of the planned trajectory in the vehicle’s coordinate system over the current and adjacent  $\tau^v$  frames.  $\mathbf{V}^v$

Methods	Language command category			Within road	User preference
	single	interaction	compound		
LCTGen [63]	0.926	0.209	0.682	0.604	0.143
ChatSim [70]	0.874	0.052	0.778	0.865	0.061
Ours	<b>0.952</b>	<b>0.883</b>	<b>0.896</b>	<b>0.935</b>	<b>0.796</b>

Table 1. High-level planning evaluation for command following rate, within road rate and user preference rate.

represents the vehicle’s centroid velocity over  $\tau^v$  frames.  $\Theta^v$  includes inherent vehicle parameters such as  $L$  (vehicle length),  $W$  (vehicle width),  $l_f$  (front overhang) and  $l_r$  (rear overhang). These parameters provide prior information that influence the physical transition process.

**Vehicle actions.** To accurately simulate real vehicle actions and maintain temporal consistency, the vehicle’s action space  $\mathcal{A}^v$  is defined by the delta steering angle  $\Delta\delta$  and scalar acceleration  $a$ , which are the two most direct controls affecting vehicle movement in actual driving.

**Vehicle transition function.** We employed the bicycle model for modeling the vehicle physical transition process. Let  $x$  and  $y$  denote the vehicle’s coordinates,  $v$  represent the scalar velocity, and  $\phi$  indicate the vehicle’s orientation. Utilizing the inherent parameters from the states, the vehicle’s state transition process can be expressed as:

$$\begin{aligned} \dot{x} &= v * \cos(\phi + \beta), & \dot{y} &= v * \sin(\phi + \beta), \\ \beta &= \arctan\left(\frac{l_r}{l_f + l_r} \tan(\delta)\right), & \dot{\phi} &= \frac{v}{l_f + l_r} \cos(\beta) \tan(\delta). \end{aligned}$$

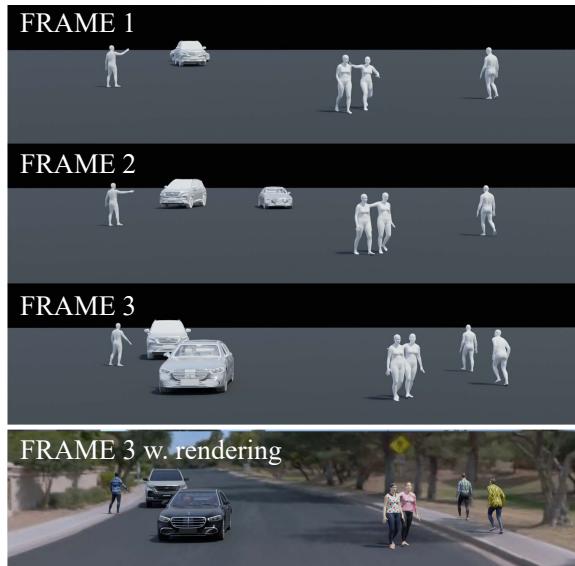
Where  $\beta$  denotes tire slip angle and  $\cdot$  denotes derivation. The bicycle model effectively simulates the physical state changes of the vehicle as determined by vehicle actions.

**Reward and training.** The reward focuses on two key aspects: following the planned trajectory and the smoothness of results. Therefore, the reward consists of two components:  $R_{pos}^v = -||\hat{p}_t - p_t||_2$  for following the planned trajectory and  $R_{act}^v = -(||\Delta\delta||_2 + ||a||_2)$  for smooth driving. Training uses TD3 [20] to maximize the accumulated discounted reward. Note that: (i) actions can be further smoothed with temporal filtering:  $\mathcal{A}_t^v = \alpha \mathcal{A}_{t-1}^v + (1 - \alpha) \mathcal{A}_{policy}^v$ , where  $\mathcal{A}_t$  is the action taken at timestep  $t$  and  $\mathcal{A}_{policy}$  is the action directly output from the policy network; (ii) obstacles can be considered by concatenating their positions and radius in the state and adding  $R_{obs}^v = \frac{\epsilon}{L_o}$  to the reward if  $L_o < D$ , where  $L_o$  is the distance to the obstacle,  $\epsilon$  is a coefficient and  $D$  is a threshold.

## 4. Experiments

### 4.1. Implementation Details and Dataset

For the PedExecutor, we utilize Isaac Gym [47] as the physics simulation engine, with the AMASS dataset [46] serving as the reference data clip for the discrimination reward. A model of the SMPL [42] robot as the simulation



**Command:** A person is taking a taxi at the left roadside, and a vehicle overtakes the taxi. Two persons are walking together with one's arm around another's shoulder. A person is chasing another person along the roadside.

Figure 5. System results under complex and composite commands, with diverse interaction information and realistic dynamics output.

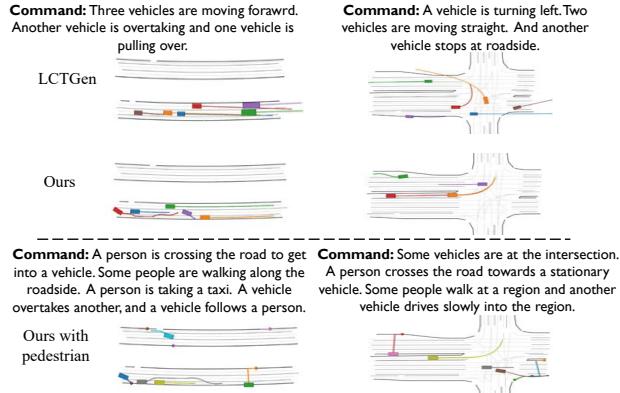
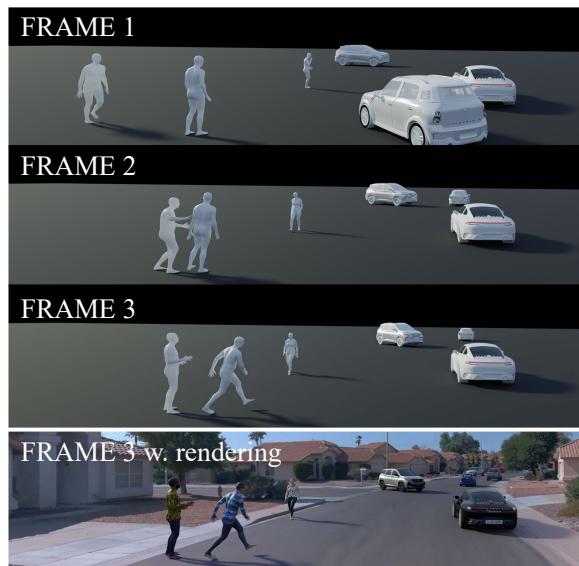


Figure 6. High-level planning results comparison under vehicle-only command, and our planning results with pedestrians involved. The boxes indicate vehicles and the circles indicate pedestrians.

object. For the VehExecutor, we construct a simulation environment incorporating physical transitions. All LLM components used in the LLM-agents are implemented with GPT-4 [1] API. For the final rendering, we followed the rendering pipeline in ChatSim [70], using Blender for rendering with SMPL based human rendering implementation. Detailed experimental settings, configurations and video results are provided in the supplementary materials.

## 4.2. System Results

We present keyframes from the output of the system in two scenes, as shown in Figure 5. Each generated output includes abundant traffic participants based on user command. From the system’s outputs, we see that: (i) interaction is thoroughly depicted, including human-vehicle in-



**Command:** A person pushes another person, and another person is making a phone call, then walks along the roadside. A vehicle turns right at the intersection, and a hurried vehicle overtakes a stationary one.

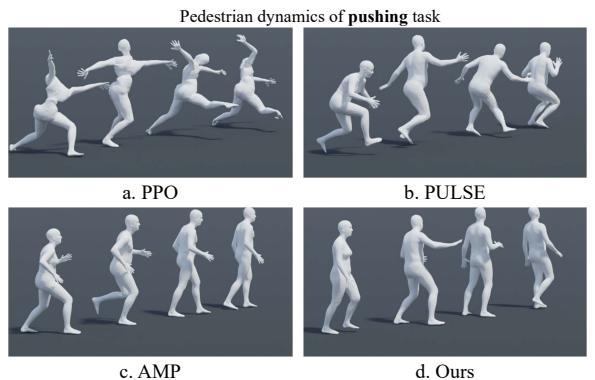


Figure 7. Visualization comparison of interaction task execution.

teractions (e.g. taking a taxi, avoidance, slowing down), vehicle-vehicle interactions (e.g. lane changes, overtaking), and human-human interactions (e.g. pushing, chasing, walking with arm around shoulder). These interactions among different types of traffic participants contribute to a more diverse and realistic scene dynamics; (ii) the system achieves precise and detailed control, with each scene description being complex, potentially including abstract semantics. Through the design of multi-LLM-agents role-playing, complex instructions are accurately analyzed and executed, and abstract semantic information is effectively broken down into executable commands for final generation; (iii) the generated results are highly realistic. The physics-based control policies for pedestrians and vehicles produce realistic, and intuitive dynamics, particularly highlighting physical feedback in human-human interaction.

Methods	FID↓	Div.↑	I-FID↓	I-Div.↑	$E_{mpjpe}\downarrow$	$E_f\downarrow$	UP↑
Pacer[56]	7.25	1.24	7.93	1.05	\	0.122	0.179
Pacer+[69]	6.62	1.58	7.76	1.28	82.33	0.128	0.289
Ours	<b>6.21</b>	<b>1.76</b>	<b>7.07</b>	<b>1.49</b>	<b>79.82</b>	0.124	<b>0.532</b>

Table 2. Dynamics quality, diversity and accuracy evaluation for trajectory following and motion imitation tasks.

Methods	Unified policy	Interaction 1	Interaction 2	Interaction 3
PPO [57]	✗	0.971	0.934	0.914
AMP [51]	✗	0.234	0.179	0.108
Pulse [45]	✗	0.975	0.942	0.925
Ours	✓	<b>0.982</b>	<b>0.977</b>	<b>0.971</b>

Table 3. Interaction tasks success rate evaluation.

### 4.3. Component Results

#### 4.3.1. LLM-agents planning

We compare the planned trajectory generated by LLM agents with existing traffic generation methods based on language, including LCTGen [63] and ChatSim [70]. Since other methods do not support scenarios involving pedestrians, we conduct an evaluation under vehicle-only instructions. We define three types of instructions: single-vehicle instructions, interaction instructions, and composite instructions. Using 5 maps, we generate 26 samples per instruction type and invite 10 users to evaluate each generated result. They assess whether each result matches the language description, whether it is within-road, and their preference for each sample. The proportion of positive responses for each criterion is summarized in Table 1. The results indicate that ChatDyn consistently produced more accurate planning results that matched descriptions and were preferred by users. We also provide visual examples in Figure 6, which also involves some planning results of ChatDyn with pedestrians. LCTGen struggles with interaction controls and complex instructions, while ChatDyn accurately fulfill all requirements, producing high-quality outputs that meet specifications, even in scenarios involving pedestrians.

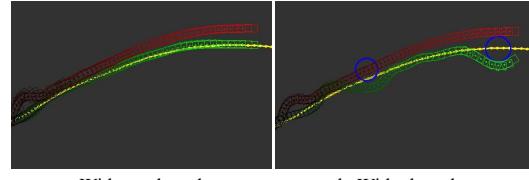
#### 4.3.2. Pedestrian executor

We follow the evaluation from Pacer+ [69] to assess trajectory following and motion imitation tasks for PedExecutor as shown in Table 2. The quality and diversity of the generated results are measured by Frechet Inception Distance (FID) [27] and diversity metric (Div.) [69] on normal speed, while I-FID and I-Div. are the metrics on low speed; imitation accuracy by Mean Per-Joint Position Error ( $E_{mpjpe}$ ), and following accuracy by following error ( $E_f$ ). Additionally, user preference (UP) is assessed by inviting 15 users to evaluate 33 segments of dynamics. With the support of hierarchical control and related training strategy, PedExecutor generates higher quality dynamics and demonstrated highly competitive performance in both following and imitation tasks.

We also evaluate the process of performing interaction

TE	HC	M. AMP	Success rate	User preference
✗	✓	✓	0.403	0.120
✓	✓	✗	0.186	0.069
✓	✗	✓	0.948	0.344
✓	✓	✓	<b>0.977</b>	<b>0.467</b>

Table 4. PedExecutor interaction task ablation study.



a. Without obstacles      b. With obstacles

Figure 8. Comparison of vehicle dynamics generation. Green boxes are our results and red boxes are Xu et al.’s [77]. Yellow lines are the planned trajectory and blue circles are obstacles.

tasks to determine each method’s effectiveness in completing these tasks. Note that PedExecutor is a unified policy handling all tasks, while other methods rely on separately trained policies for each specific interaction task. Three types of interaction tasks are selected for training and validation: pushing (interaction 1), patting (interaction 2), and walking with an arm around the shoulder (interaction 3). The experimental results are presented in Table 3, with visual results shown in Figure 7. From the both results, we see that, AMP [51] generates dynamics resembling the reference motion, but struggles to access suitable reference data clips for interaction tasks, resulting in actions that only superficially mimic the reference without fulfilling the task requirements. PPO [57] and PULSE [45] demonstrate relatively high task success rates, but the visualizations show that their outputs lack human-like style and appear unnatural. In contrast, with hierarchical control and body masked AMP, PedExecutor achieves both high success rates and maintains high-quality, human-like outputs.

As shown in Table 4, we also conduct ablation studies on interaction tasks to verify the effectiveness of task embedding (TE), hierarchical control (HC), and body masked AMP (M. AMP). The results indicate that, without task embedding, PedExecutor struggles to accurately identify the required interaction task, leading to difficulty in successful task completion. When body masked AMP is removed, similar to AMP, the discrimination reward lacks a task-completing reference. In this situation, the outcomes can merely be close to the reference data clip without the ability to successfully execute the intended task. Without hierarchical control, the absence of action priors slows the training process and reduces the success rates. More video comparison results can be found in the supplementary materials.

#### 4.3.3. Vehicle executor

We evaluate the accuracy of VehExecutor under varying initial speeds by measuring the position error and velocity

Methods/Speed	0	5	10	20
PP [10]	0.129/0.103	0.143/0.125	0.162/0.142	0.231/0.208
Xu et al. [77]	0.075/0.054	0.084/0.066	0.095/0.077	0.138/0.114
Ours	<b>0.059/0.037</b>	<b>0.062/0.041</b>	<b>0.077/0.054</b>	<b>0.106/0.088</b>

Table 5. Position/velocity error of vehicle dynamics generation.

Filt.	Action spa.	His. state	Position error	Velocity error
×	×	×	0.138	0.114
✓	×	×	0.132	0.111
✓	✓	×	0.115	0.092
✓	✓	✓	<b>0.106</b>	<b>0.088</b>

Table 6. VehExecutor ablation study.

error, comparing our approach with Xu et al.’s method [77] and pure pursuit [10] (PP) in Table 5. VehExecutor consistently achieves the best performance across all speeds. We also conduct an ablation study in Table 6, examining the effects of action filtering (Filt.), the composition of the action space (Action spa.) which means it consists directly of velocity and steering or their variations, and the inclusion of historical state (His. state) information. The ablation results validate the effectiveness of each design. Additionally, visual experiments in Figure 8 further demonstrate our method achieves more precise tracking, and obstacle avoidance capabilities in scenarios with obstacles.

## 5. Conclusion and Limitations

We propose ChatDyn, the first system to achieve interactive and realistic language-driven multi-actor dynamics generation in street scenes. ChatDyn utilizes multi-LLM-agent role-playing to enable interaction-aware high-level planning under instructions. We introduce PedExecutor, a unified control policy for realistic pedestrian dynamics generation across multiple tasks, also enabling fine-grained interactions. We introduce VehExecutor, a physics-based vehicle control policy with a history-aware design to generate realistic vehicle dynamics. The dynamics output can be used as a prior for various kinds of simulators including renderer, reconstruction, and generation model.

**In the future**, we plan to integrate more types of dynamics with diverse characteristics, such as cyclists and certain common animals, to reflect more realistic and varied scene dynamics. At the same time, more complex and diverse interaction behaviors can be further explored to construct scene information and events with greater accuracy. The relevant methodologies, after specific optimizations, can also be applied to more closed, yet finer-grained indoor scenes.

## References

- [1] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al.
- [2] Harley Amado, Sara Ferreira, Jose Pedro Tavares, Paulo Ribeiro, and Elisabete Freitas. Pedestrian–vehicle interaction at unsignalized crosswalks: a systematic review. *Sustainability*, 12(7):2805, 2020. 1
- [3] Nikos Athanasiou, Mathis Petrovich, Michael J Black, and GÜl Varol. Teach: Temporal action composition for 3d humans. In *2022 International Conference on 3D Vision (3DV)*, pages 414–423. IEEE, 2022. 2
- [4] Jinseok Bae, Jungdam Won, Donggeun Lim, Cheol-Hui Min, and Young Min Kim. Pmp: Learning to physically interact with environments using part-wise motion priors. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–10, 2023. 2
- [5] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023. 3
- [6] Luca Bergamini, Yawei Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Błażej Osiński, Hugo Grimmett, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5119–5125. IEEE, 2021. 3
- [7] Michael J Black, Priyanka Patel, Joachim Tesch, and Jinlong Yang. Bedlam: A synthetic dataset of bodies exhibiting detailed lifelike animated motion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8726–8737, 2023. 2
- [8] Holger Caeser, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liang, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1
- [9] He Chen, Hongpinng Ren, Rui Li, Guang Yang, and Shanshan Ma. Generating autonomous driving test scenarios based on openscenario. In *2022 9th International Conference on Dependable Systems and Their Applications (DSA)*, pages 650–658. IEEE, 2022. 3
- [10] R Craig Coulter. Implementation of the pure pursuit path tracking algorithm. *Technical report, DTIC Document*, 1992. 9, 3
- [11] Rishabh Dabral, Muhammad Hamza Mughal, Vladislav Golyanik, and Christian Theobalt. Mofusion: A framework for denoising-diffusion-based motion synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9760–9770, 2023. 2
- [12] Wenzhun Dai, Ling-Hao Chen, Jingbo Wang, Jinpeng Liu, Bo Dai, and Yansong Tang. Motionlcm: Real-time controllable motion generation via latent consistency model. *arXiv preprint arXiv:2404.19759*, 2024. 2
- [13] Wenhao Ding, Yulong Cao, Ding Zhao, Chaowei Xiao, and Marco Pavone. Realgen: Retrieval augmented generation for controllable traffic scenarios. *arXiv preprint arXiv:2312.13303*, 2023. 3

- [14] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017. 1
- [15] Zhiyang Dou, Xuelin Chen, Qingnan Fan, Taku Komura, and Wenping Wang. C-ase: Learning conditional adversarial skill embeddings for physics-based characters. In *SIGGRAPH Asia 2023 Conference Papers*, pages 1–11, 2023. 2
- [16] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. 3
- [17] Lan Feng, Quanyi Li, Zhenghao Peng, Shuhan Tan, and Bolei Zhou. Trafficgen: Learning to generate diverse and realistic traffic scenarios. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3567–3575. IEEE, 2023. 3
- [18] Yao Feng, Jing Lin, Sai Kumar Dwivedi, Yu Sun, Priyanka Patel, and Michael J. Black. ChatPose: Chatting about 3D human pose. In *CVPR*, 2024. 3
- [19] Daniel J Fremont, Tommaso Dreossi, Shromona Ghosh, Xiangyu Yue, Alberto L Sangiovanni-Vincentelli, and Sanjit A Seshia. Scenic: a language for scenario specification and scene generation. In *Proceedings of the 40th ACM SIGPLAN conference on programming language design and implementation*, pages 63–78, 2019. 3
- [20] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018. 6
- [21] Luiz G Galvão and M Nazmul Huda. Pedestrian and vehicle behaviour prediction in autonomous vehicle system—a review. *Expert Systems with Applications*, 238:121983, 2024. 1
- [22] Team GLM, Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Dan Zhang, Diego Rojas, Guanyu Feng, Hanlin Zhao, et al. Chatglm: A family of large language models from glm-130b to glm-4 all tools. *arXiv preprint arXiv:2406.12793*, 2024. 3
- [23] Mahsa Golchoubian, Moojan Ghafurian, Kerstin Dautenhahn, and Nasser Lashgarian Azad. Pedestrian trajectory prediction in pedestrian-vehicle mixed environments: A systematic review. *IEEE Transactions on Intelligent Transportation Systems*, 2023. 1
- [24] Mahir Gulzar, Yar Muhammad, and Naveed Muhammad. A survey on motion prediction of pedestrians and vehicles for autonomous driving. *IEEE Access*, 9:137957–137969, 2021. 1
- [25] Chuan Guo, Shihao Zou, Xinxin Zuo, Sen Wang, Wei Ji, Xingyu Li, and Li Cheng. Generating diverse and natural 3d human motions from text. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5152–5161, 2022. 2
- [26] Chuan Guo, Yuxuan Mu, Muhammad Gohar Javed, Sen Wang, and Li Cheng. Momask: Generative masked modeling of 3d human motions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1900–1910, 2024. 2, 5
- [27] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017. 8
- [28] Sirui Hong, Xiawu Zheng, Jonathan Chen, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, et al. Metagpt: Meta programming for multi-agent collaborative framework. *arXiv preprint arXiv:2308.00352*, 2023. 3
- [29] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *ACM Transactions on Software Engineering and Methodology*, 2023. 3
- [30] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021. 3
- [31] Stefan Jesenski, Jan Erik Stellet, Florian Schiegg, and J Marius Zöllner. Generation of scenes in intersections for the validation of highly automated driving functions. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 502–509. IEEE, 2019. 3
- [32] Jordan Juravsky, Yunrong Guo, Sanja Fidler, and Xue Bin Peng. Padl: Language-directed physics-based character control. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, 2022. 2
- [33] Jihoon Kim, Jiseob Kim, and Sungjoon Choi. Flame: Free-form language-based motion synthesis & editing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 8255–8263, 2023. 2
- [34] Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvuri, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*, 2024. 3
- [35] Yan Leng and Yuan Yuan. Do llm agents exhibit social behavior? *arXiv preprint arXiv:2312.15198*, 2023. 3
- [36] Binxu Li, Tiankai Yan, Yuanting Pan, Jie Luo, Ruiyang Ji, Jiayuan Ding, Zhe Xu, Shilong Liu, Haoyu Dong, Zihao Lin, et al. Mmedagent: Learning to use medical tools with multimodal agent. *arXiv preprint arXiv:2407.02483*, 2024. 3
- [37] Junkai Li, Siyu Wang, Meng Zhang, Weitao Li, Yunghwei Lai, Xinhui Kang, Weizhi Ma, and Yang Liu. Agent hospital: A simulacrum of hospital with evolvable medical agents. *arXiv preprint arXiv:2405.02957*, 2024. 3
- [38] Weiwei Li and Emanuel Todorov. Iterative linear quadratic regulator design for nonlinear biological movement systems. In *First International Conference on Informatics in Control, Automation and Robotics*, pages 222–229. SciTePress, 2004. 3
- [39] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36, 2024. 3

- [40] Weiyang Liu, Zeju Qiu, Yao Feng, Yuliang Xiu, Yuxuan Xue, Longhui Yu, Haiwen Feng, Zhen Liu, Juyeon Heo, Songyou Peng, Yandong Wen, Michael J. Black, Adrian Weller, and Bernhard Schölkopf. Parameter-efficient orthogonal finetuning via butterfly factorization. In *ICLR*, 2024. 3
- [41] Xiao Liu, Hao Yu, Hanchen Zhang, Yifan Xu, Xuanyu Lei, Hanyu Lai, Yu Gu, Hangliang Ding, Kaiwen Men, Kejuan Yang, et al. Agentbench: Evaluating llms as agents. *arXiv preprint arXiv:2308.03688*, 2023. 3
- [42] Matthew Loper, Naureen Mahmood, Javier Romero, Gerard Pons-Moll, and Michael J Black. Smpl: A skinned multi-person linear model. In *Seminal Graphics Papers: Pushing the Boundaries, Volume 2*, pages 851–866, 2023. 6, 2
- [43] Jack Lu, Kelvin Wong, Chris Zhang, Simon Suo, and Raquel Urtasun. Scenecontrol: Diffusion for controllable traffic scene generation. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 16908–16914. IEEE, 2024. 3
- [44] Zhengyi Luo, Jinkun Cao, Kris Kitani, Weipeng Xu, et al. Perpetual humanoid control for real-time simulated avatars. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10895–10904, 2023. 2, 1
- [45] Zhengyi Luo, Jinkun Cao, Josh Merel, Alexander Winkler, Jing Huang, Kris Kitani, and Weipeng Xu. Universal humanoid motion representations for physics-based control. *arXiv preprint arXiv:2310.04582*, 2023. 2, 5, 8
- [46] Naureen Mahmood, Nima Ghorbani, Nikolaus F Troje, Gerard Pons-Moll, and Michael J Black. Amass: Archive of motion capture as surface shapes. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5442–5451, 2019. 6
- [47] Viktor Makovychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021. 6
- [48] Ying Ni, Menglong Wang, Jian Sun, and Keping Li. Evaluation of pedestrian safety at intersections: A theoretical framework based on pedestrian-vehicle interaction patterns. *Accident Analysis & Prevention*, 96:118–129, 2016. 1
- [49] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 3
- [50] Xianghe Pang, Shuo Tang, Rui Ye, Yuxin Xiong, Bolun Zhang, Yanfeng Wang, and Siheng Chen. Self-alignment of large language models via monopolylogue-based social scene simulation. *arXiv preprint arXiv:2402.05699*, 2024. 3
- [51] Xue Bin Peng, Ze Ma, Pieter Abbeel, Sergey Levine, and Angjoo Kanazawa. Amp: Adversarial motion priors for stylized physics-based character control. *ACM Transactions on Graphics (ToG)*, 40(4):1–20, 2021. 2, 4, 5, 8
- [52] Xue Bin Peng, Yunrong Guo, Lina Halper, Sergey Levine, and Sanja Fidler. Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters. *ACM Transactions On Graphics (TOG)*, 41(4):1–17, 2022. 2, 1
- [53] Zeju Qiu, Weiyang Liu, Haiwen Feng, Yuxuan Xue, Yao Feng, Zhen Liu, Dan Zhang, Adrian Weller, and Bernhard Schölkopf. Controlling text-to-image diffusion by orthogonal finetuning. In *NeurIPS*, 2023. 3
- [54] Rodrigo Queiroz, Thorsten Berger, and Krzysztof Czarnecki. Geoscenario: An open dsl for autonomous driving scenario representation. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pages 287–294. IEEE, 2019. 3
- [55] Davis Rempe, Jonah Phlion, Leonidas J Guibas, Sanja Fidler, and Or Litany. Generating useful accident-prone driving scenarios via a learned traffic prior. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17305–17315, 2022. 3
- [56] Davis Rempe, Zhengyi Luo, Xue Bin Peng, Ye Yuan, Kris Kitani, Karsten Kreis, Sanja Fidler, and Or Litany. Trace and pace: Controllable pedestrian animation via guided trajectory diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13756–13766, 2023. 1, 2, 8
- [57] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 5, 8
- [58] Yonatan Shafir, Guy Tevet, Roy Kapon, and Amit H Bermano. Human motion diffusion as a generative prior. *arXiv preprint arXiv:2303.01418*, 2023. 2
- [59] Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueteng Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *Advances in Neural Information Processing Systems*, 36, 2024. 3
- [60] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 1, 3
- [61] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10400–10409, 2021. 3
- [62] Shuhan Tan, Kelvin Wong, Shenlong Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Scenegen: Learning to generate realistic traffic scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 892–901, 2021. 3
- [63] Shuhan Tan, Boris Ivanovic, Xinshuo Weng, Marco Pavone, and Philipp Kraehenbuehl. Language conditioned traffic generation. *arXiv preprint arXiv:2307.07947*, 2023. 1, 3, 6, 8, 4
- [64] Wei Tao, Yucheng Zhou, Yanlin Wang, Wenqiang Zhang, Hongyu Zhang, and Yu Cheng. Magis: Llm-based multi-agent framework for github issue resolution. *arXiv preprint arXiv:2403.17927*, 2024. 3
- [65] Chen Tessler, Yoni Kasten, Yunrong Guo, Shie Mannor, Gal Chechik, and Xue Bin Peng. Calm: Conditional adversarial

- latent models for directable virtual characters. In *ACM SIGGRAPH 2023 Conference Proceedings*, pages 1–9, 2023. 2
- [66] Sourabh Thakur and Subhadip Biswas. Assessment of pedestrian-vehicle interaction on urban roads: a critical review. *Archives of transport*, 51, 2019. 1
- [67] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. 3
- [68] Weilin Wan, Zhiyang Dou, Taku Komura, Wenping Wang, Dinesh Jayaraman, and Lingjie Liu. Tlcontrol: Trajectory and language control for human motion synthesis. *arXiv preprint arXiv:2311.17135*, 2023. 2
- [69] Jingbo Wang, Zhengyi Luo, Ye Yuan, Yixuan Li, and Bo Dai. Pacer+: On-demand pedestrian animation controller in driving scenarios. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 718–728, 2024. 1, 2, 5, 8
- [70] Yuxi Wei, Zi Wang, Yifan Lu, Chenxin Xu, Changxing Liu, Hao Zhao, Siheng Chen, and Yanfeng Wang. Editable scene simulation for autonomous driving via collaborative llm-agents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15077–15087, 2024. 6, 7, 8, 2, 4
- [71] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. Physics-based character controllers using conditional vaes. *ACM Transactions on Graphics (TOG)*, 41(4):1–12, 2022. 2
- [72] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155*, 2023. 3
- [73] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, et al. Pandaset: Advanced sensor suite dataset for autonomous driving. In *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*, pages 3095–3101. IEEE, 2021. 1
- [74] Yiming Xie, Varun Jampani, Lei Zhong, Deqing Sun, and Huaizu Jiang. Omnicontrol: Control any joint at any time for human motion generation. *arXiv preprint arXiv:2310.08580*, 2023. 2
- [75] Pei Xu, Xiumin Shang, Victor Zordan, and Ioannis Karamouzas. Composite motion learning with task control. *ACM Transactions on Graphics (TOG)*, 42(4):1–16, 2023. 2
- [76] Pei Xu, Kaixiang Xie, Sheldon Andrews, Paul G Kry, Michael Neff, Morgan McGuire, Ioannis Karamouzas, and Victor Zordan. Adaptnet: Policy adaptation for physics-based character control. *ACM Transactions on Graphics (TOG)*, 42(6):1–17, 2023. 2
- [77] Yinda Xu and Lidong Yu. Drl-based trajectory tracking for motion-related modules in autonomous driving. *arXiv preprint arXiv:2308.15991*, 2023. 8, 9, 3
- [78] Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*, 2023. 3
- [79] Heyuan Yao, Zhenhua Song, Baoquan Chen, and Libin Liu. Controlvae: Model-based learning of generative controllers for physics-based characters. *ACM Transactions on Graphics (TOG)*, 41(6):1–16, 2022. 2
- [80] Mingyuan Zhang, Zhongang Cai, Liang Pan, Fangzhou Hong, Xinying Guo, Lei Yang, and Ziwei Liu. Motondiffuse: Text-driven human motion generation with diffusion model. *arXiv preprint arXiv:2208.15001*, 2022. 2
- [81] Mingyuan Zhang, Xinying Guo, Liang Pan, Zhongang Cai, Fangzhou Hong, Huirong Li, Lei Yang, and Ziwei Liu. Re-modiffuse: Retrieval-augmented motion diffusion model. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 364–373, 2023. 2
- [82] Ziyuan Zhong, Davis Rempe, Yuxiao Chen, Boris Ivanovic, Yulong Cao, Danfei Xu, Marco Pavone, and Baishakhi Ray. Language-guided traffic simulation via scene-level diffusion. In *Conference on Robot Learning*, pages 144–177. PMLR, 2023. 1, 3
- [83] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3560–3566. IEEE, 2023. 3
- [84] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023. 3
- [85] Wenyang Zhou, Zhiyang Dou, Zeyu Cao, Zhouyingcheng Liao, Jingbo Wang, Wenjia Wang, Yuan Liu, Taku Komura, Wenping Wang, and Lingjie Liu. Emdm: Efficient motion diffusion model for fast and high-quality motion generation. In *European Conference on Computer Vision*, pages 18–38. Springer, 2025. 2

# ChatDyn: Language-Driven Multi-Actor Dynamics Generation in Street Scenes

## Supplementary Material

### S1. Details of PedExecutor

#### S1.1. PedExecutor State Details

The components of the humanoid's proprioception  $S_{prop}^p$  are as follows: joint positions  $\mathbf{j} \in \mathbb{R}^{24 \times 3}$ , rotations  $\mathbf{q} \in \mathbb{R}^{24 \times 6}$ , linear velocities  $\mathbf{v} \in \mathbb{R}^{24 \times 3}$ , and angular velocities  $\omega \in \mathbb{R}^{24 \times 3}$  [69]. These components are normalized with respect to the agent's heading and root position in our simulator. The rotation  $\mathbf{q}$  is represented using a 6-degree-of-freedom rotation representation.  $S_{prop}^p$  along with the trajectory state  $S_{traj}^p$ , the motion state  $S_{mo}^p$  to be mimicked, and the target state  $S_{tar}^p$  for interaction, collectively forms the complete observation. During the task masking process, the remaining relevant states are masked by multiplying them by 0 based on the task to be executed, and in the motion state, specific joints can also be masked by multiplying them by 0 if necessary. For example, in the experiment, only the upper-body related states were retained.

#### S1.2. Task-related Reward and Training Details

**Reward designs.** For trajectory following tasks [56], the reward is defined as  $R_{\text{trajectory}} = e^{-2\|\hat{p}_t - p_t\|}$ , where  $\hat{p}_t$  is the position to follow at  $t$ ,  $p_t$  is the current character position. For motion imitation tasks [44],  $R_{\text{imitation}} = e^{-100\|\hat{j}_{pos}^t - j_{pos}^t\| \odot m^t} + e^{-10\|\hat{j}_{rot}^t - j_{rot}^t\| \odot m^t} + e^{-0.1\|\hat{j}_{vel}^t - j_{vel}^t\| \odot m^t} + e^{-0.1\|\hat{j}_{\omega}^t - j_{\omega}^t\| \odot m^t}$ , where  $\hat{\cdot}$  indicates the motion states to be imitated,  $m^t$  is the mask to select the joints for imitation, which are the joints of upper body in our experiments. Different interaction tasks require distinct reward designs [52]; here, we present three sample tasks: for pushing,  $R_{\text{pushing}} = 1 - u^{up} \cdot u$ ,  $u^{up}$  is the global up vector  $u^*$  is the target's up vector; for patting,  $R_{\text{patting}} = e^{-\|p^{rh} - c\|}$ ,  $p^{rh}$  is the position of right hand and  $c$  is the target contact position; and for walking while bending the shoulder,  $R_{\text{walking.bending}} = e^{-2\|\hat{p}_t - p_t\|} + e^{-\|p^{rh} - c\|}$ , which combines the trajectory following and target position contacting. The final reward is calculated as  $R = 0.5 \cdot R_{\text{disc}} + 0.5 \cdot R_{\text{task}}$ .

**Interaction process.** The specific execution process for the three interaction tasks can be understood as follows: i) pushing, where the goal is to push the interaction object over, ii) patting, where the task is to gently tap a specific part of the interaction object (e.g., the shoulder) without knocking it over, and iii) walking with arm around another's shoulder, where the agent walks along a specific path while keeping the arm in contact with a specific location on the interaction object (e.g., the shoulder). During the training of the interaction tasks, the interaction object is replaced with

a box in the physical environment to facilitate more stable training. However, during testing, the interaction object is replaced with another character in the simulation environment, and physical collisions and interactions are present, leading to the final output in the test phase.

**Failure recovery for robustness.** To enhance the robustness of PedExecutor, specifically to ensure that it can handle external disturbances without failing due to perturbations caused by physical collisions, we incorporate recovery during its training [44]. In this approach, the human pose is initialized in a collapsed or otherwise unstable standing state. Training is then conducted from this initial state, allowing the policy to learn how to recover from failure. This enables the policy to exhibit robustness when responding to physical collisions and interactions. Without this, the policy might fail to continue the action after even minor disturbances.

### S1.3. Network Architecture

All relevant models, including the baselines, adopt the same network architecture, using an MLP as the policy network with hidden layers of 2048 and 1024 units. The final output is directed to either the latent space or the PD control signal, depending on whether hierarchical control is employed. The remaining network components, such as the discriminator, value network, control frequency (30Hz), and hyperparameters used for training, are consistent with those adopted in Pacer+ [69]. All training and testing are conducted on an NVIDIA 4090.

### S1.4. Evaluation Details

**Following and imitation.** For the following and imitation tasks, we adopt the same computation methods as those used in Pacer+ [69]. The calculations of FID and diversity are performed using the same manual feature extraction approach as in Pacer+, with 1000 segments selected from the AMASS dataset for FID computation. For the low-speed 1-FID and 1-diversity, we also follow Pacer+ by testing on instances where the speed is less than 1 m/s.

**Interaction tasks.** For the three interaction-related tasks: i) pushing, the success criterion is whether the object is pushed over within the specified timestep (with a tilt along the z-axis greater than 30°), and no part of the body other than the hands is in contact with the target; ii) patting, the success criterion is whether, within the specified timestep, the right hand is within 5 cm of the target's specific location and remains there for at least 50 timesteps,

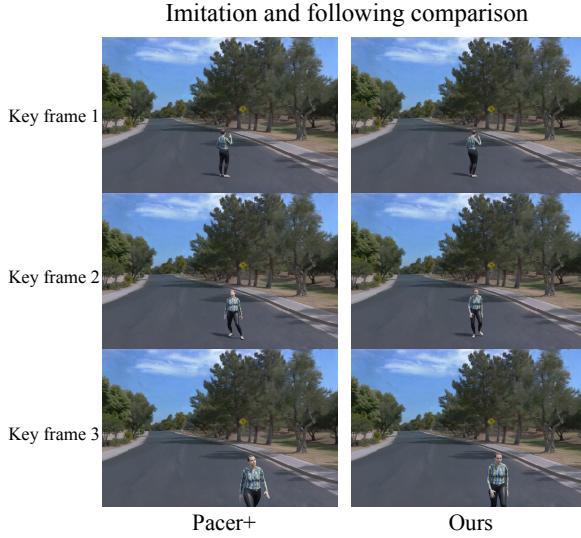


Figure S1. Comparison of imitation and following.

with no other part of the body in contact with the target except the hands; iii) walking with arm around another's shoulder, the success criterion is that the maximum deviation from the reference trajectory is no greater than 10 cm, and the right hand is within 5 cm of the target's specific location for at least 150 timesteps. In the interaction ablation study, for the user study, we recruit 15 participants to test 30 dynamic sequences and select the ones they considered to have the highest quality.

### S1.5. Rendering Details

We utilize the rendering pipeline from ChatSim [70], employing Blender's Cycles as the rendering engine. Background rendering and HDRI lighting from ChatSim are applied to the scene. The human model is based on SMPL [42], with the corresponding mesh initialized, and rendered using skin and clothing textures provided in the Bedlam dataset [7].

### S1.6. Supplementary Experiments

We provide more comprehensive visualization results to compare different aspects of PedExecutor's characteristics. All comparisons, along with dynamic results, can also be observed in the supplementary video material.

**Imitation and following comparison.** As shown in S1, we compared the visual effects of Pacer+ during the imitation and following processes. It is evident that, under the influence of hierarchical control, PedExecutor enables smoother and more seamless transitions between different tasks—transitions that cannot be achieved with the priors provided by AMP.

**Results of imitation during following.** As shown in S2,

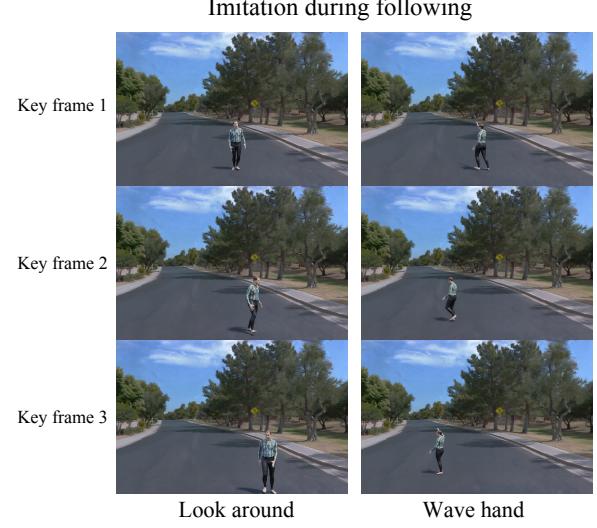


Figure S2. Imitation during following.

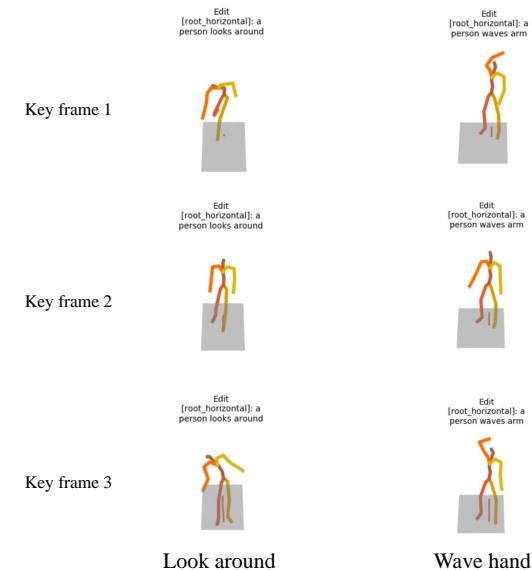


Figure S3. Failure of PriorMDM [58] with action specification during following.

we also provide the results of PedExecutor performing both imitation and following simultaneously. In the case of following a specific trajectory, the upper body performs actions such as looking around and waving, showcasing capability of PedExecutor to handle both imitation and following tasks concurrently.

**Interaction task comparison.** As shown in S4 S5 S6, we compared the performance of AMP, PPO, PULSE, and our method across three interaction tasks. It is still evident that while PPO and PULSE are able to complete the

tasks under the given conditions, they fail to produce natural and human-like results. AMP, on the other hand, can only approximate the reference in the discriminator (walking or running), and is completely unable to complete the tasks. In contrast, our PedExecutor successfully completes the tasks while generating natural and human-like results.

### S1.7. Further Discussion of Kinematics Methods

For kinematics-based methods, some approaches can achieve following and motion specification, but they are completely unable to handle interaction-related tasks. Additionally, for following and motion specification tasks, these methods often suffer from overfitting to the dataset, leading to suboptimal performance. As shown in S3, the results in noticeable sliding steps and unnatural movements.

## S2. Details of VehExecutor

### S2.1. Network Architecture and Parameters of Bicycle-model

All networks in VehExecutor are implemented as MLPs. The policy network consists of layers with dimensions 256, 256, 128, 128, 64, and 64, while the value network has layers with dimensions 1024, 512, 256, and 128. During training, the parameters of the bicycle model ( $L$ ,  $W$ ,  $l_f$ ,  $l_r$ ) are set to two configurations: (2.7, 1.8, 0.9, 0.9) and (6.1, 2.5, 2.3, 2.0), mixed to accommodate vehicles of varying sizes. These parameters can be adjusted as needed based on specific requirements, with the two configurations provided here serving as examples.

### S2.2. Obstacle state

The states of obstacles are composed of their orientation and distance relative to the vehicle's own coordinate system (considering the radius of the obstacles). The state vector is initialized with a maximum number of observable obstacles. The vector is then populated with the specific identifiers of the actual obstacles, and any remaining entries are masked. Obstacles that are too far away are directly excluded, meaning their states are not considered, and they do not contribute to the reward calculation. The distance threshold for exclusion is set to 10 in the experiment.

### S2.3. Supplementary Experiments

**Robustness.** We also provide the results using LQR [38] in S1. It can be observed that LQR is capable of vehicle dynamics generation from a planned trajectory to some extent. However, the planned trajectory lacks relevant constraints, which may lead to unreasonable turns or abrupt changes, causing LQR to often produce less than ideal results. Furthermore, we tested the robustness of vehicle dynamics generation by adding noise with a mean of 0 and variance of  $\sigma$  to the planned trajectory. As shown in S2, the results show

Methods-Speed	0	5	10	20
LQR [38]	0.074/0.058	0.086/0.070	0.092/0.079	0.125/0.0108

Table S1. Position/velocity error of LQR.

$\sigma$	PP [10]	Xu et al. [77]	LQR [38]	ours
0.00	0.162/0.142	0.095/0.077	0.092/0.079	<b>0.077/0.054</b>
0.01	0.168/0.147	0.098/0.080	0.322/0.289	<b>0.079/0.058</b>
0.03	0.169/0.151	0.098/0.079	0.568/0.479	<b>0.082/0.060</b>

Table S2. Vehicle dynamics generation under Gaussian noise.  $\sigma$  indicates the variance of noise.

that LQR is highly sensitive to noise, often producing significantly worse outcomes under its influence, while other methods are relatively less affected by the noise.

**Visualization for effectiveness.** In the supplementary video, we provide results comparing the planned trajectories without using VehExecutor and with VehExecutor. It is evident that, without the involvement of VehExecutor, the dynamics generated by simply calculating heading between consecutive frames of the directly planned trajectory are highly unnatural, exhibiting noticeable tail swings and abrupt changes. In contrast, the results using VehExecutor are much more realistic and natural. This demonstrates the necessity of VehExecutor, as trajectories without physical constraints are highly unnatural and impractical.

## S3. Details of High-level Planning

### S3.1. LLM-agent details

We provide the relevant sample prompts for the LLM agent in S7 and S8. All outputs from the LLM are in JSON format, and corresponding follow-up functions are used to convert the JSON outputs into the required data structures.

Similar to the rendering process, we also use the Waymo Open Dataset [60] as the planning dataset in all experiments, and the final results are presented based on this dataset.

### S3.2. Different LLMs

We further validated the impact of different LLMs on the results in S3. We conducted experiments using GPT-3.5 [49] and Llama-3 [16] 70B (smaller models struggle to accurately execute the instructions). All other experimental settings remained consistent with those in the main text. It is evident that while other LLMs can handle the task to some extent, GPT-4 [1] demonstrates the most accurate understanding and decomposition of the instructions.

### S3.3. Collision handling

In the high-level planning process, we also designed a collision handling mechanism to avoid unintended collisions.

Methods	Language command category			Within road
	single	interaction	compound	
Ours-Llama3	0.885	0.742	0.812	0.920
Ours-GPT3.5	0.854	0.738	0.834	0.915
Ours-GPT4	<b>0.952</b>	<b>0.883</b>	<b>0.896</b>	<b>0.935</b>

Table S3. High-level planning evaluation for different LLMs.

	ChatSim [70]	LCTGen [63]	Ours
Collision rate	0.149	0.092	0.067

Table S4. Collision rate of high-level planning

Specifically, during the trajectory generation, collision detection is performed, and when a collision is detected, a velocity adjustment function is applied to modify the speed of one of the agents. This velocity adjustment function uses a nonlinear mapping to combine the original planned result with an interpolated trajectory, leading to a planning result with different speeds. We evaluated the probability of collisions across 50 generated samples, which is calculated by dividing the number of vehicles that experienced a collision by the total number of vehicles. As shown in S4, the collision rate for all methods remains low, and ChatDyn also achieves a low collision rate while incorporating collision handling.

## Pushing

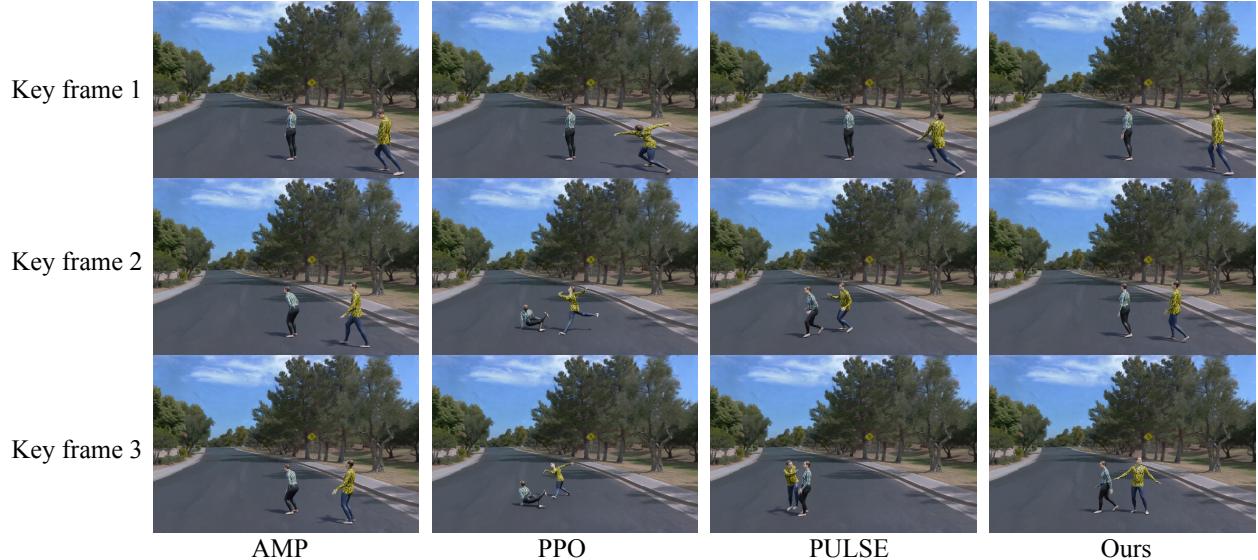


Figure S4. Comparison of pushing.

## Patting



Figure S5. Comparison of patting.

### Walking with arm around another's shoulder



Figure S6. Comparison of walking with arm around another one's shoulder.

### Oracle agent prompt.

I have a requirement for analyzing a scenario. I will provide you with a requirement, and I need your help to break it down into four pieces of information: (1) identify all the agents, (2) initialize each agent's state, (3) formulate a text for each agent. You should provide the information in JSON format. Note that your output should only include the JSON format of the information, not the analysis process.

(1) Identify all the agents: This means you need to extract all the agents from the scenario based on the input text. The text may be quite lengthy, so you can locate the agents by identifying the nouns, which may assist you in this task. The agents must be the objects involved in autonomous driving. The format should be as follows: `agent_list = {'0': 'agent_name_0', '1': 'agent_name_1', ...}`. The keys (e.g., '0', '1', etc.) represent unique agent IDs, which you should assign starting from '0'. The agents should only include vehicles like cars, pedestrians, trucks, and buses, and should not include static objects like trees or buildings. Ensure that each agent is given a distinct name. For example, in the sentence 'car a wants to overtake car b', the nouns are 'car a' and 'car b'. Both are objects in autonomous driving scenarios. Therefore, the `agent_list` should be formatted as follows: `agent_list = {'0': 'car_a', '1': 'car_b'}`. Pedestrians can be represented with identifiers like 'ped\_0' and may share numbering with entities such as 'car\_0'.

(2) Initialize each agent's state. In this task, you need to determine four aspects for each agent: 1. Agent type, 2. Movement, 3. Speed. You can use the results from task (1) to complete this task. Provide the initial state of each agent in a list, formatted in JSON. Most time the speed is bigger than 0. For example, the initial states should be defined as follows: `init_states = ['agent_id': '0', 'agent_type': 'car', 'movement': 'overtaking', 'speed': 60, 'agent_id': '1', 'agent_type': 'car', 'movement': 'straight', 'speed': 30]`. If one car intends to overtake another, it should ideally be at least twice as fast as the car it is overtaking. The initial state must include the agent type, movement, and speed.

Agent type should in [pedestrian, vehicle], action should in ['static', 'straight', 'pull over', 'turn over', 'overtake', 'turn left', 'turn right', 'straight left', 'straight right'] if it's a vehicle, and in ['static', 'crossing', 'straight'] if it is a pedestrian.

(3) Formulate a text for each agent. As an omniscient observer, you should instruct each agent on their actions through a text. The text should contain two pieces of information: 1. The agent type, 2. The agent's intention. You need to provide this in a JSON format. `guide_texts = '0': 'text1', '1': 'text2', '2': 'text3', ...` where the key is the agent's ID and the value is the text. The agent's name should be consistent with those in the `agent_list`.

Your answer should be in a JSON format, and must include the three information: `agent_list, init_state, guide_texts`. And `init-state` must include the agent type, movement and speed.

Figure S7. Oracle agent prompt.

### Actor agent prompt.

Now, you are an agent in the autonomous driving scenario. I will give you a text, and agent\_list, describing who you are and what you need to do. Note that your output should not include your analysis process, only the JSON format of the information you provide.

I need you to analyze the text and give me the four information of the ego agent to describe what type of lane the agent should be in: (1) depend (2) speed change (3) keypoints list (4) behavior. Note that you only need to give the information of the ego agent, not the other agents.

(1) depend. You need to determine the depend of the agent according to the agent's intention. And give it like [depend\_agent\_id, depend\_type]. You can get the depend\_agent\_id from the agent\_list. Depend\_type should in ['end', 'start', 'trajectory', 'None']. 'end' represents the ego agent's end point is the depend agent's end point, 'start' represents the ego agent's start point is the depend agent's start point, 'trajectory' represents the ego agent's whole trajectory is the depend agent's start point. If it has no depend, you should give it [-1, 'None'].

(2) speed change. You need to determine the speed change of the agent according to the agent's intention. If you want to speed up, the speed change should be 1. If you want to slow down, the speed change should be -1. If you want to keep the speed, the speed change should be 0. For example, If there is someone nearby, you should slow down.

(3) keypoints list. You need to determine the number of keypoints required for your current behavior, the confirmation method for each keypoint, and their respective parameters, then return a list containing this information. Each keypoint can be confirmed in one of three ways: (1) Map-based (type '0'), with parameters: lane position ('left', 'right', or 'front'), lane type ('centerline' or 'boundary'), and driving direction ('turn left', 'turn right', or 'straight'). (2) Lane-relationship-based (type '1'), with the parameter being the relationship to the previous keypoint ('opposite direction adjacent', 'same direction adjacent', 'adjacent straight', 'adjacent left turn', 'adjacent right turn', 'different type adjacent', or 'opposite boundary'). (3) Agent-based (type '2'), with parameters specifying required information type ('point' or 'trajectory'). The result should be a sequential list of dictionaries where the key is the type ('0', '1', or '2') and the value is the corresponding parameters. For example, a left-turning car might return '[0: 'position': 'front', 'lane\_type': 'centerline', 'direction': 'turn left', '1: 'relationship': 'adjacent straight']', while a car in the left lane picking up 'ped\_a' might return '[0: 'position': 'left', 'lane\_type': 'centerline', 'direction': 'straight', '2: 'info\_type': 'point']'. Ensure compliance with parameters, common sense, and traffic regulations, with the first keypoint typically using type '0'.

(4) behavior. You need to determine your behavior. If you are a vehicle, the behavior is "None." If you are a pedestrian, the behavior corresponds to the description provided. There are two scenarios: if your behavior is a specific action such as calling or waving, simply return the text of that action; if the behavior involves interactive actions such as pushing, patting, or walking with an arm around another person, you must return the exact predefined descriptions for these three types of interactions without modification. For example, if you are calling, your behavior is "calling phone.;" if you push ped\_1, your behavior is "pushing."

Your answer should be in a JSON format, and must include depend, speed change, keypoints list and behavior.

Figure S8. Actor agent prompt.