# *Text-to-Drive*: Diverse Driving Behavior Synthesis via Large Language Models

Phat Nguyen[1], Tsun-Hsuan Wang[2], Zhang-Wei Hong[2], Sertac Karaman[3], Daniela Rus[2]

***Abstract*— Generating varied scenarios through simulation is crucial for training and evaluating safety-critical systems, such as autonomous vehicles. Yet, the task of modeling the trajectories of other vehicles to simulate diverse and meaningful close interactions remains prohibitively costly. Adopting language descriptions to generate driving behaviors emerges as a promising strategy, offering a scalable and intuitive method for human operators to simulate a wide range of driving interactions. However, the scarcity of large-scale annotated language-trajectory data makes this approach challenging. To address this gap, we propose Text-to-Drive (T2D) to synthesize diverse driving behaviors via Large Language Models (LLMs). We introduce a knowledge-driven approach that operates in two stages. In the first stage, we employ the embedded knowledge of LLMs to generate diverse language descriptions of driving behaviors for a scene. Then, we leverage LLM's reasoning capabilities to synthesize these behaviors in simulation. At its core, T2D employs an LLM to construct a state chart that maps low-level states to high-level abstractions. This strategy aids in downstream tasks such as summarizing low-level observations, assessing policy alignment with behavior description, and shaping the auxiliary reward, all without needing human supervision. With our knowledge-driven approach, we demonstrate that T2D generates more diverse trajectories compared to other baselines and offers a natural language interface that allows for interactive incorporation of human preference. Please check our website for more examples: here**

## I. INTRODUCTION

Simulators have emerged as an effective tool for training and evaluating safety-critical systems, such as autonomous vehicles. They provide opportunities to synthesize novel data for training, expose methods to edge cases that are otherwise not available in public driving datasets, and offer a cost-effective method of simulating close interactions that are otherwise costly or impractical to capture in real-world settings. Their utility extends further to in-simulation validations and enables detailed studies that are difficult to observe directly.

Despite their advantages, current simulators face significant challenges in controlling the behaviors of surrounding vehicles and scaling these interactions. Adding varied driving behaviors to the simulation can facilitate comprehensive testing across diverse driving behavior profiles. This addresses the inherent bias found in driving datasets, used by data-driven simulators, which tend to be limited in scope and curated from a narrow selection of geographic areas.

One promising direction to overcome these limitations is by extending the capabilities of foundational models into simulators. A knowledge-driven approach that utilizes the embedded knowledge of Large Language Models (LLMs) to curate comprehensive and diverse driving scenarios, eliminates the need for exhaustive manual scripting of potential interactions. Furthermore, utilizing natural language to control the generation of these scenarios presents an intuitive method to specify desired behavior trajectories. This technique allows for generating driving scenarios based on language descriptions, making it easier for human operators to curate meaningful test cases. A notable application of this strategy involves connecting to textual data, such as accident reports, to ground simulations in real-world contexts. Our work adopts this knowledge-driven approach, drawing on rich knowledge sources to generate diverse driving scenarios. This method can be complementary to data-driven simulators, which rely only on human-driving data.

[1]UMass Amherst, [2]MIT CSAIL, [3]MIT LIDS.

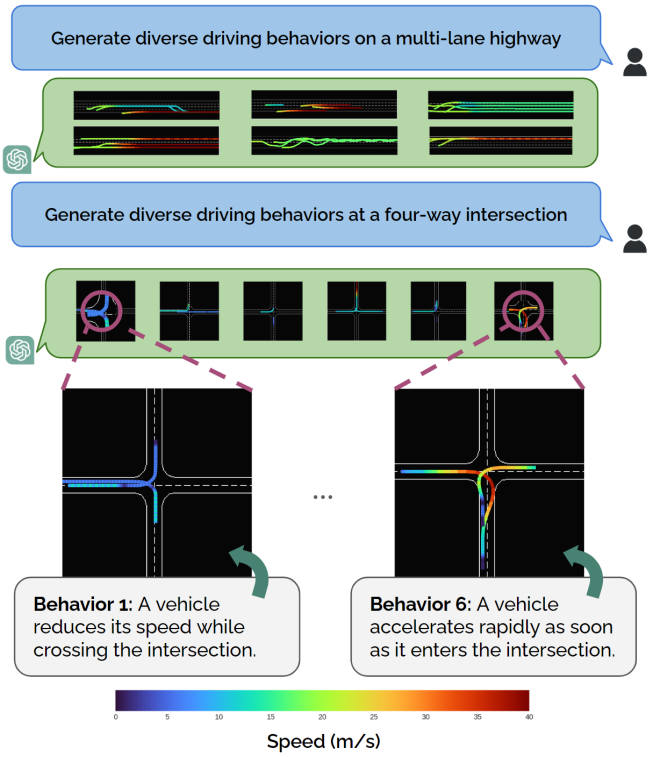



Fig. 1: Given a scene description, ***T2D*** leverages Large Language Models to generate diverse descriptions of driving behaviors and then synthesizes them in simulation.

In this paper, our research aims to answer the question: *How can we translate a diverse set of language descriptions of driving behaviors into a corresponding set of behaviorally diverse policies for simulation?* To address this, we introduce ***T2D***, a knowledge-driven method for simulation that utilizes LLMs to generate diverse language descriptions of driving behaviors and then synthesizes them in simulation. Given

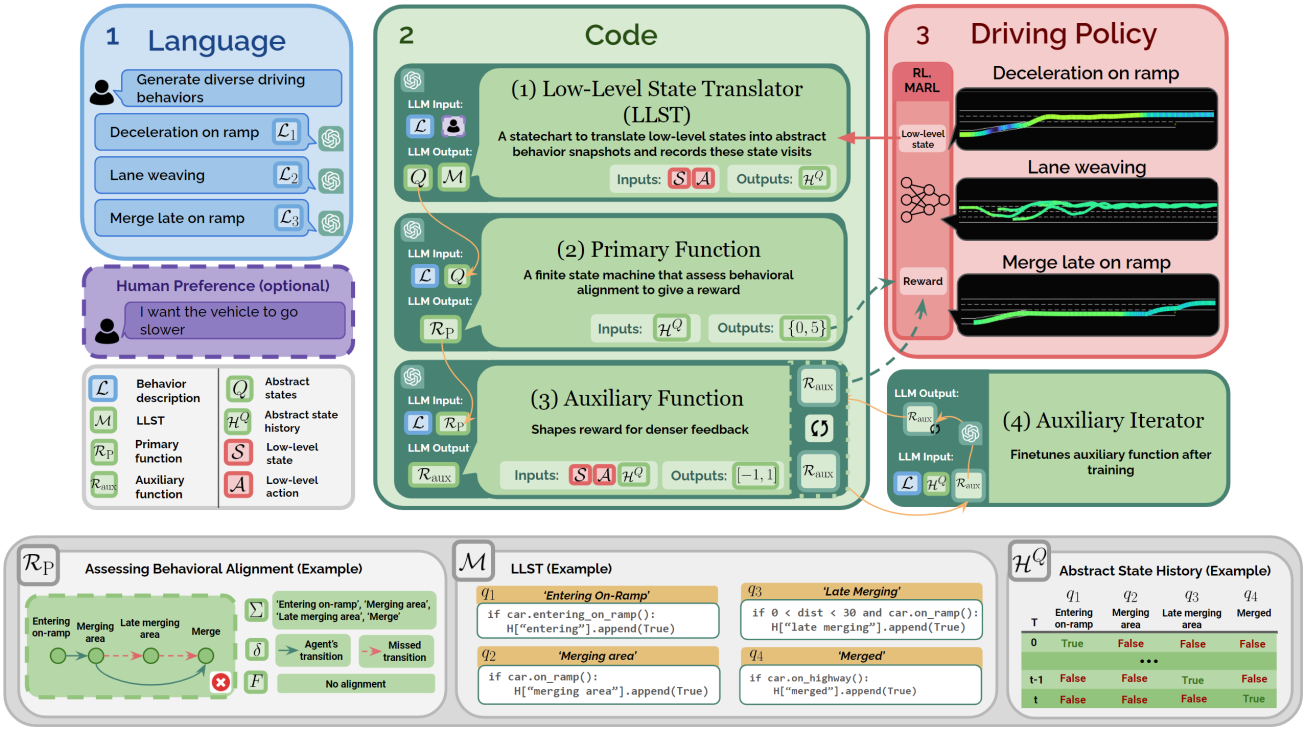



Fig. 2: **Overview.** *Left*: First, an LLM generates diverse descriptions of driving behaviors, which can incorporate human preferences through a natural language interface. *Middle*: Next, an LLM generates a low-level state translator (LLST), primary function, and auxiliary function from a description of a driving behavior. The LLST translates low-level states to abstract states (see example in *bottom middle* block) and then records their state visit history (see example in *bottom right* block). The primary function gives a reward only when the vehicle exhibits the target behavior, using a finite-state machine for formal verification of behavior emergence (see example in *bottom left* block). The auxiliary function provides rewards for reaching intermediate states and can be iteratively updated. *Right*: Finally, we employ a standard multi-agent RL framework to train a driving policy using the primary and auxiliary functions as guidance.

a behavior description, ***T2D*** generates a mapping of low-level states (e.g: vehicle position, heading, speed) to high-level abstractions (e.g. "on the on-ramp", "near the end of on-ramp", and "merged"). By leveraging this abstract state representation, transitions are defined to capture the temporal dynamics of the behavior, effectively embodying temporal logic. Such a framework not only enhances the capacity for reasoning about behavioral alignment but also creates abstract summaries of low-level observations. The ability to assess the behavioral alignment is used as a primary reward function to guide the driving policy, while our auxiliary function is used to improve exploration efficiency. The abstract summaries inform the LLM whether and how to adjust the auxiliary function after training. This iterative process introduces new incentives for exploring new states and penalties for unwanted behaviors, thereby aligning the policy more closely with the desired behavior. We demonstrate that ***T2D*** maintains the behavioral context across natural language, code, and driving policy, enabling accurate simulation of the driving behavior. Additionally, ***T2D*** surpasses baselines in generating diverse trajectories and offers a natural language interface to embed human preferences into the driving trajectories. Using ***T2D***, we generated 18 driving behaviors from language descriptions. To this end, we make the following key contributions:

- We introduce ***T2D***, a knowledge-driven method for simulation that enables (i) text-to-driving behavior synthesis, and (ii) diverse driving behavior generation.
- Our method facilitates the use of LLM-based reasoning by encapsulating the logic in state machines. This facilitates complex policy training processes such as: (i) summarizing low-level observations, (ii) reasoning about behavioral alignment, and (iii) iteratively updating the auxiliary function, without any human supervision.
- We demonstrate our method effectively retains the behavioral context across natural language, code, and driving policy, enabling it to simulate a driving behavior from a description. Additionally, ***T2D*** not only generates more diverse trajectories compared to baselines but also offers a language interface to integrate human preferences into driving simulations.

## II. RELATED WORK

### A. Driving Simulators

Collecting data for specific scenarios proves challenging, emphasizing the need to test safety-critical systems in controlled environments. This necessity has spurred the development of model-based simulators [8], capable of modeling real-world physics and constructing photorealistic environments. Continual improvements by the CARLA

open-source development community have integrated tools like Scenic [13], enabling the creation of complex traffic scenarios using compact English-like syntax. However, an issue with early simulators such as CARLA is the sim-to-real domain gap. In contrast, data-driven simulators like VISTA [1] and LIDARSim [19] have shown potential in bridging this gap by leveraging real-world driving data [4], [26] to reconstruct real-world scenes and synthesize novel views. These simulators, however, lack the generative control that model-based simulators offer. Our work addresses this issue by extending the generative capabilities of foundation models into simulators, specifically those that are equipped with a reinforcement learning (RL) interface [14], [16].

### B. Behavior Generation

**Diverse Skills.** Unsupervised skill acquisition algorithms such as DIAYN [10] and DADS [24] have demonstrated their ability to learn diverse skills in unsupervised settings. These have further inspired methods aimed at enhancing trajectory diversity in RL [17] and simulating diverse traffic behaviors [25]. However, despite their ability to generate diverse actions, their methods cannot be directed through textual guidance or incorporate human preference.

**Controllable Traffic Generation.** Recent advancements in data-driven traffic generation methods have employed neural networks to synthesize new scenarios [2], [27], [22], [11], [29], [30]. These approaches offer a promising avenue for the realistic modeling of traffic environments. Other works have focused on learning latent representations of scenarios for querying, editing, and composing driving behaviors from driving datasets [7]. Reinforcement learning with human feedback has also been explored to incorporate human preference to generated scenarios [5]. Despite their ability to create novel scenes, these methods cannot take language descriptions of driving behaviors as inputs. Additionally, generative approaches using diffusion models have also been studied [6], [31] as a suitable interactive traffic simulation of safety-critical scenarios. The necessity of manual labeling for driving behaviors in recent research [7] further highlights this limitation. Our knowledge-driven approach aims to overcome this issue by enabling the generation of diverse driving behaviors from rich knowledge sources.

More recently, language-conditioned traffic generation has been explored in [28], [31], [23]. [28] leverages LLMs to translate textual descriptions of traffic scenes directly into driving trajectories. However, while their approaches focuses on low-level trajectory generation, ours explores the generation of diverse high-level behaviors such as "tailgating" and "lane weaving". In this work, we build upon the capabilities of LLMs for zero-shot generation of reward functions [18]. Our research explores this capability further and extends it to diverse driving behaviors for simulation, especially in scenarios lacking a ground-truth fitness function.

## III. METHOD

The goal of our method is to generate diverse driving behaviors from textual inputs through a knowledge-driven approach, structured into two main stages. In the first stage, we utilize the vast knowledge embedded in LLMs to generate language descriptions of diverse driving behaviors (Sec. III-A). In the second stage, as we transition from these descriptions to driving policies, three key steps are undertaken. First, we used an LLM to generate a low-level state translator (Sec. III-C). We detail how this is used to translate low-level trajectories to an abstract code representation. Then, to evaluate the alignment of driving policies with desired behaviors, we introduce the primary reward function (Sec. III-D), which, together with our auxiliary function, provides reward guidance to the driving policy (Sec. III-E). Subsequently, we detail an iterative approach that employs LLMs to adjust the auxiliary function after each training. Finally, we employ a multi-agent RL framework to train a driving policy (Sec. III-G). An overview of our method is shown in Figure 2.

### A. Generating Behavior Descriptions

In our knowledge-driven approach, we use `gpt-4`'s zero-shot generation capability to generate descriptions of diverse driving behaviors, $\mathcal{L} \sim \pi^L(.|\text{scene})$ from a concise description of a scene. For each scene type – intersections, merges, highways – we generated 50 descriptions of driving behaviors. We then selected a smaller, representative subset of six behaviors per scene for simulation to provide a detailed analysis of all generated behaviors.

### B. Retrieval from Environment

We enhance our code generation model, $\pi^C$, with retrieval augmented generation (RAG) to provide sufficient context about the simulation environment. This involves segmenting the source code using an abstract syntax tree (AST), embedding the code using a text embedding model (`text-embedding-ada-002`), and storing the embeddings in a database. To preserve the dependencies between the code segments, we use ctags to generate a repository map. We used LangChain to implement our RAG framework.

During the retrieval process, we use the behavior description $\mathcal{L}$ to query the embedding database and retrieve relevant code segments. We use this code and a repository map as context for $\pi^C$. By making the source code accessible, we enrich the code generation model with APIs about the low-level state. For instance, in addressing the behavior "accelerative merging on-ramp", the model can utilize the attribute `car.speed` and access functions `car.on_ramp()`. This retrieval ensures that the low-level observations of the driving policy are available in the code space.

### C. Low-Level State Translator (LLST)

Given a behavior description $\mathcal{L}$, our code generation model $\pi^C$, generates a low-level state translator, $\mathcal{M} \sim \pi^C(.|\mathcal{L})$. The state translator $\mathcal{M}$, has three primary responsibilities: it (1) decomposes the behavior, (2) maps lower-level states to abstract states, and (3) records abstract state visits. First, $\mathcal{M}$ decomposes the behavior into abstract states, $Q$. Each abstract state, $q \in Q$, captures an essential aspect of the driving behavior. For example, in the case of "merging late

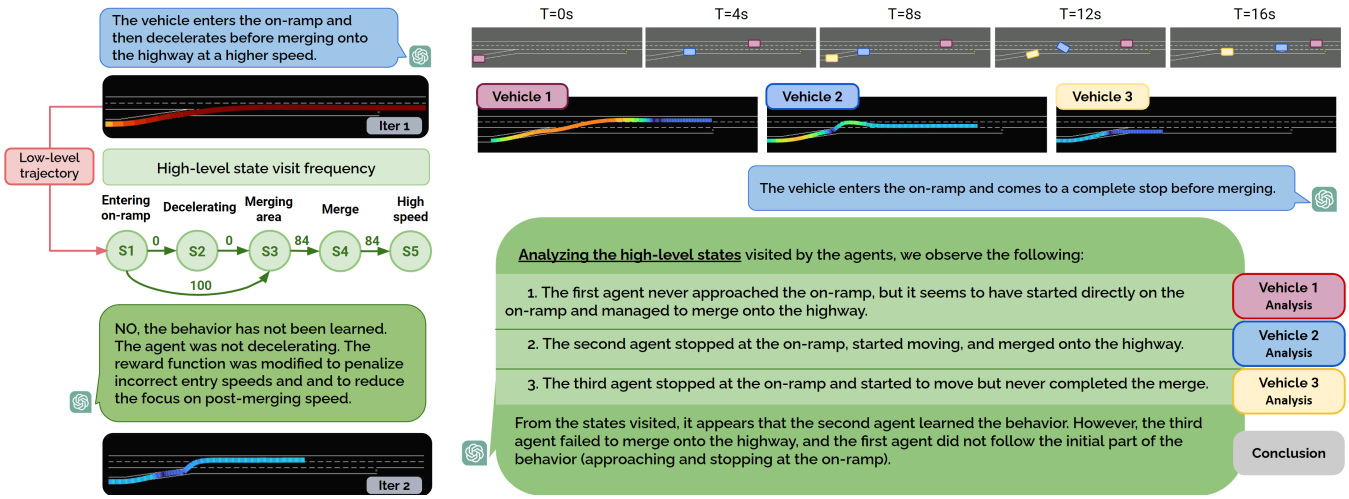



Fig. 3: *Left*: The auxiliary iterator LLM analyzes the policy after training to decide whether and how to adjust the auxiliary function based on the history of abstract state visits. *Right*: The right figure illustrates the LLM's reasoning process, where it reads a high-level behavior sequence, analyzes it, and then provides an accurate summary of the low-level trajectories.

on the on-ramp", $q$ could be any of "on the on-ramp," "merging," and "near the end of the on-ramp" as illustrated in Figure 2. This decomposition yields four advantages: firstly, it discretizes the behavior into critical phases to capture different aspects of the behavior; secondly, it constrains $Q$ to be relevant to the target behavior; thirdly, by merging the initial advantages, it imparts clear, objective guidance which makes the generation of $\mathcal{M}$ more consistent and reliable; and lastly, it allows the state name to be in the language domain.

Moreover, $\mathcal{M}$ is constructed by the LLM as a state chart. We define $\mathcal{M}$ as a tuple $\mathcal{M} = (Q, \mathcal{T}, E, U, G)$, where $\mathcal{T}$ is the set of transitions triggered by an event $E$, conditioned on a guard in $G$, and results in an update action from $U$. The events in $E$ represent low-level changes within the driving environment, such as speed, position, and heading. The guards in $G$ are boolean functions that return true under certain conditions in the low-level state. We exploit the code-generating capabilities of LLMs to achieve semantic alignment between guard conditions and abstract state names. This step leverages the code-generation model, $\pi^C$, with our RAG framework. Specifically, we used the `gpt-4-1106-preview` variant [21] and set the temperature to 0.2. This decision aims for a more deterministic output due to the objective nature of the task.

The translator's update action from $U$, results in an update in the state history dictionary $\mathcal{H}^Q$, that keeps a historical record of all abstract state visits. With each timestep, it appends a boolean value indicating whether a state has been visited. The state history dictionary $\mathcal{H}^Q$ over a rollout of $T$ timesteps can be represented as $\mathcal{H}^Q : Q \rightarrow \{\text{true}, \text{false}\}^T$. Then $\mathcal{H}^Q(q) = \mathcal{H}^q$, is the visit history associated with $q$. *This enables both the ability to identify current state occupancy and track state visits and transitions.* These characteristics are extremely effective at summarizing low-level observations of the driving policy back to the code and language space (see Figure 3) which we utilized in the primary function (Sec. III-D) and iterator (Sec. III-F).

### D. Primary Reward Function

In this section, we introduce the primary function $\mathcal{R}_\text{P}$, generated by an LLM, which takes $\mathcal{H}^Q$ as input and returns a reward, $\mathcal{R}_\text{P} : \mathcal{H}^Q \rightarrow \{0, 5\}$. This function serves two purposes: first, it assesses the behavioral alignment of the driving policy $\pi^P$ with the target behavior $\mathcal{L}$; second, it awards a large reward when the vehicle demonstrates the target behavior. To generate $\mathcal{R}_\text{P} \sim \pi^C(.|(Q, \mathcal{L}))$, we combine the abstract state names $Q$, and behavior description $\mathcal{L}$, as inputs to $\pi^C$. The LLM constructs $\mathcal{R}_\text{P}$ as a finite-state machine (FSM) that models the target behavior $\mathcal{L}$. This FSM can be described as a tuple, $\mathcal{R}_\text{P} = (Q, \Sigma, \delta, q_0, F)$; where $\Sigma = \{q | q \in Q\}$ is the input alphabet, $\delta : Q \times \Sigma \rightarrow Q$ is the transition function, $q_0$ is the initial state, and $F$ is the accepting states, indicating the target behavior is achieved.

The abstract state of $\mathcal{R}_\text{P}$ utilizes $Q$ from the state translator, $\mathcal{M}$. We regard each $q \in Q$ from $\mathcal{M}$ as a code abstraction representing a behavior snapshot. For instance, the state "end of on-ramp" is an abstraction for the conditions $0 <$ `car.headway()` $< 30$ and `car.on_ramp()`. The behavioral transition function $\delta$, then adds transitions between abstract states to capture the temporal dynamics of the target behavior. This is particularly useful for driving behaviors such as "late merging", which requires visiting "end of on-ramp" before transitioning to the "merge" state. The formal structure of the FSM, generated by the LLM, provides a framework for verifying the abstract behavior sequences given by $\mathcal{H}^Q$. This strategy encapsulates the LLM's reasoning into a compact FSM that can be accessed after LLM inference. This structured format *enables an offline application that utilizes LLM's reasoning to assess behavioral alignment during the training of the driving policy*. For a visual illustration of the FSM, refer to Figure 2. In our implementation, we set the environment minimum speed to 0 so that the vehicle could not reverse to avoid complex generation of $\mathcal{R}_\text{P}$.

We then use the FSM to give a reward of 5 to the vehicle

upon reaching the accepted states, $F$. This reason will be evident in the next section (Sec. III-E). An important nuance of our method is its bi-functional relationship between the state translator and the primary function. The state translator abstracts low-level observations of the driving policy into the code domain, while $\mathcal{R}_{\text{P}}$ evaluates these abstractions in the code domain and gives a reward to guide the driving policy $\pi^P$. This reciprocal relationship encourages behavioral consistency across different spaces.

### E. Auxiliary Reward Function

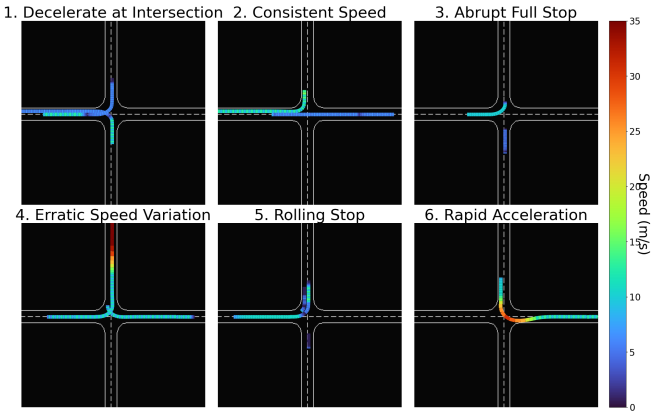
Although the primary function $\mathcal{R}_{\text{P}}$, is used to guide the driving policy $\pi^P$, the rewards from $\mathcal{R}_{\text{P}}$ are too sparse. To improve exploration efficiency, we use an auxiliary function, $\mathcal{R}_{\text{aux}} : \mathcal{S}, \mathcal{A}, \mathcal{H}^Q \rightarrow [-1, 1]$, that takes low-level state $\mathcal{S}$, action $\mathcal{A}$, and abstract state history $\mathcal{H}^Q$, as inputs and returns a reward. The purpose of providing more input to the auxiliary function is to enable reward guidance for both low-level and abstract states. Our auxiliary function is generated using the code generation model $\pi^C$, with $\mathcal{R}_{\text{P}}$ and $\mathcal{L}$ as inputs, $\mathcal{R}_{\text{aux}} \sim \pi^C(.|(\mathcal{R}_{\text{P}}, \mathcal{L}))$. We use the same RAG framework and LLM model with a temperature of $0.7$ to allow for more creative reward shaping.

The auxiliary function purpose is to provide denser rewards for reaching intermediate states, while simultaneously injecting the behavioral context into the driving policy. This is particularly useful for complex behaviors characterized by numerous transitions where the primary reward function offers limited guidance. For instance, the behavior "merging from a complete stop" requires the vehicle to navigate a series of actions: enter the on-ramp, stop, move again, and then merge. Using the auxiliary reward, we successfully trained 18 driving behaviors (see Section III-G).

The rewards from our $\mathcal{R}_{\text{aux}}$ are normalized to the range $[-1, 1]$. Empirically, we find that as the driving policy learns the behavior, the auxiliary function's influence naturally diminishes as the primary function's larger rewards become more dominant. This makes our method more robust to different generations of the auxiliary function.

### F. Auxiliary Function Iterator

A common challenge of reward shaping is that it can generate unintended behaviors. To address this, we generate a new iteration of the auxiliary function using a code generation model, $\mathcal{R}'_{\text{aux}} \sim \pi^C(.|(\mathcal{R}_{\text{aux}}, \mathcal{H}^q, \mathcal{L}))$; where $\mathcal{R}'_{\text{aux}}$ is the new auxiliary function, and $(\mathcal{R}_{\text{aux}}, \mathcal{H}^q, \mathcal{L})$ are the inputs to the LLM. The abstract state history $\mathcal{H}^q$, as an input to $\pi^C$, provides an informative abstract summary of the low-level trajectories. This summary provides the LLM with high-level insights into a rollout, allowing it to adjust the reward incentive structure accordingly. This iterative process introduces new incentives for exploring new states and penalties for unwanted behaviors, thereby aligning the policy more closely with the desired behavior. By incorporating high-level observations into the iterative process, we proactively mitigate risks associated with unsafe reward-shaping practices [15]. For a visual illustration of this process, see the left figure in Figure 3.




| Behaviors | Emergence Rate (%) | Collision Rate (%) | Avg. Speed (m/s) |
|---|---|---|---|
| 1. Decelerate through intersection | 56.67 | 40.00 | 7.42 |
| 2. Consistent speed crossing | 63.33 | 20.00 | 5.73 |
| 3. Abrupt full stop at intersection | 56.67 | 13.33 | 1.06 |
| 4. Erratic speed | 93.33 | 26.67 | 5.60 |
| 5. Rolling stop at intersection | 73.33 | 30.00 | 5.10 |
| 6. Rapid acceleration at intersection | 76.67 | 43.33 | 21.24 |

Fig. 4: Diverse driving behaviors at an intersection.

### G. Learning a Driving Policy.

We employed a multi-agent implementation of the Advantage Actor-Critic algorithm (MAA2C) [20] to learn the driving policies $\pi^P$. In this setup, each agent independently learns using a concurrent training strategy in a cooperative environment under partial observation conditions.

## IV. EXPERIMENTS

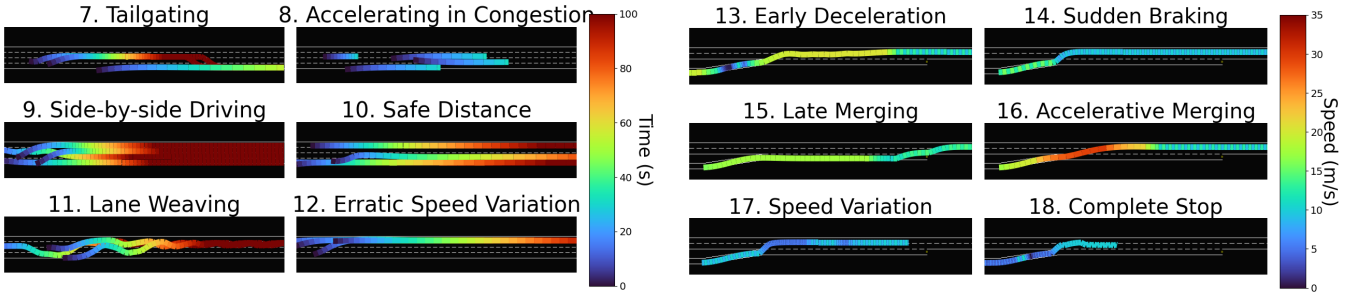
In this section, we demonstrate our method's ability to preserve the behavioral context across natural language, code, and driving policy by showcasing strong alignment between these domains. Following this, we propose a suite of metrics to quantify the behavioral diversity in code and driving policy. Through evaluation, we show that our method can generate more diverse trajectories than other baselines.

### A. Implementation Details

**Simulator.** We conducted training using the Highway Environment simulator [16]. We simulate for 100 timesteps and update the policy at a frequency of 5Hz. The action space for each vehicle is discrete, comprising 5 possible actions for lateral and longitudinal control with a speed range of $[0, 40]$. Our implementation uses the OpenAI Gym framework [3].

**Training Details.** Within the MARL framework, we shared rewards for vehicles nearby and penalized for collisions. All policies were trained using the same hyperparameters, network architecture, and environment setup. Specifically, our actor and critic networks each comprise two hidden linear layers, each with 256 neurons followed by a ReLU activation function. We used the RMSprop optimizer and applied a fixed learning rate of $5e^{-5}$ for both networks.

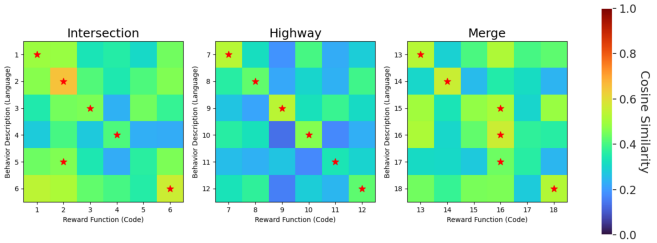
**Iterating Details.** We train a driving policy $\pi^P$, for $10,000$ episodes and do a soft evaluation for every $2,500$



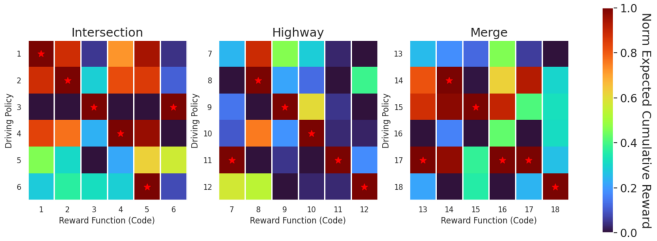

| Behaviors | Emergence Rate (%) | Collision Rate (%) | Avg. Speed (m/s) |
|---|---|---|---|
| Tailgating | 40.00 | 56.67 | 19.62 |
| Accelerating in Congestion | 100.00 | 73.33 | 30.18 |
| Side-by-Side Driving | 63.33 | 33.33 | 14.28 |
| Following at a Safe Distance | 93.33 | 0.00 | 31.79 |
| Lane Weaving | 73.33 | 36.67 | 15.52 |
| Erratic Speed | 53.00 | 63.33 | 29.96 |

| Behaviors | Emergence Rate (%) | Collision Rate (%) | Avg. Speed (m/s) |
|---|---|---|---|
| Early Deceleration on Ramp | 80.00 | 30.00 | 12.46 |
| Sudden Braking After Merging | 80.00 | 13.33 | 9.79 |
| Late Merging at Ramp End | 86.67 | 0.00 | 13.17 |
| Accelerative Merging | 60.00 | 33.33 | 12.43 |
| Merging with Speed Variation | 96.67 | 6.67 | 8.72 |
| Merging from Complete Stop | 60.00 | 26.67 | 9.05 |

Fig. 5: Diverse highway driving and merging behaviors.




(a) A diagonal line in the code and language agreement matrix indicates that there is a high similarity between the language description and code, and thus we show that the behavioral context is preserved across these domains.




(b) A diagonal line in the code and driving policy agreement matrix indicates that the policy trained by the reward function was most optimal compared to the other evaluated policies, and therefore we show that the behavioral context is preserved across these domains.

Fig. 6: Agreement matrix to show behavioral alignment.

episodes. This uses our iterative process to assess the alignment of $\pi^P$ to the behavior $\mathcal{L}$ without updating the auxiliary function. We terminate the training if the iterator indicates the behavior has been learned. After training for $10,000$ episodes, we run the iterative process again and update the auxiliary function according to the iterator.

**Evaluation Details.** We evaluate the driving policies on 30 rollouts (varied seeding) using the policy with the highest expected cumulative reward during training.

### *B. Policy Alignment*

In this section, we demonstrate behavioral alignment between the language, code, and driving policy domains using agreement matrices, and validate text-to-driving policy synthesis via manual human inspection.

**Language and Code Agreement.** To quantify the agreement between the language and code domain, we compute the pairwise cosine similarity between the sentence embedding of $\mathcal{L}$ and the code embedding of $\mathcal{R}_{\text{aux}}$ using CodeBERT [12]. For a set of $n$ language descriptions and auxiliary functions, our agreement matrix is a $n \times n$ matrix. A diagonal line in the agreement matrix indicates that the correct pairing of $\mathcal{L}$ and $\mathcal{R}_{\text{aux}}$ received the highest similarity among other possible $\mathcal{L}$ and $\mathcal{R}_{\text{aux}}$ pairs. Figure 6a presents the visualization of the agreement matrix, where a diagonal line is mostly present. Notably, the figure for the "highway" environment distinctly shows a pronounced bright diagonal, surrounded by darker regions, indicating a strong alignment. This may be from the greater diversity in language descriptions of the highway behaviors compared to those of other environments.

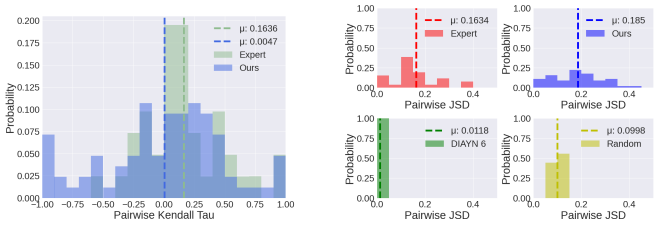
**Code and Driving Policy Agreement.** In addition, we extend our analysis to also show strong agreement between code and driving policy domain. To this end, we quantify this agreement by evaluating the trained driving policy $\pi^P$ and computing the expected cumulative reward according to the auxiliary reward $\mathcal{R}_{\text{aux}}$. Specifically, given a set of $n$ auxiliary functions and $n$ driving policies, we define the element $\mathbf{A}_{ij}^{C \leftrightarrow P}$ of the agreement matrix $\mathbf{A}^{C \leftrightarrow P} \in \mathbb{R}^{n \times n}$ as:

$$\mathbf{A}_{ij}^{C \leftrightarrow P} = \mathbb{E}_{\tau \sim \pi_j^P} \left[ \sum_{t=1}^{T} \mathcal{R}_i(\mathcal{S}_t, \mathcal{A}_t) \right] \tag{1}$$

Then, we compare these values relative to the performance of alternate driving policies on the same auxiliary reward. Formally, we consider a reward function $\mathcal{R}_i$ to be in agreement with a policy $\pi_j^P$ if:

$$\forall k \neq j, \mathbf{A}_{ij}^{C \leftrightarrow P} > \mathbf{A}_{ik}^{C \leftrightarrow P} \tag{2}$$

The intuition is to examine how each driving policy is evaluated by the auxiliary function. The presence of a diagonal line in $\mathbf{A}^{C \leftrightarrow P}$, seen in Figure 6b suggests strong alignment between the code and driving policy domain. As observed again, the "highway" environment showcases a prominent dark red diagonal that is surrounded by darker blue areas. This implies that the policies are highly specialized and are, therefore, behaviorally diverse. Our results indicate that




(a) Code diversity with $K = 5$ (b) Policy diversity

Fig. 7: Pairwise diversity comparison.

natural language descriptions may serve as a proxy for estimating the diversity of driving behaviors.

**Language and Policy Agreement.** In our next analysis, we verify that the driving policy $\pi^P$ corresponds to the behaviors described by $\mathcal{L}$. To evaluate the degree of alignment between the behavior description and the driving policy, we enlist human annotators to measure the emergence rate by manual inspection. The emergence rate measures how frequently the described behavior appears by at least 1 agent in 30 different rollouts. The relatively high emergence rates, as collectively presented in Section III-G, strongly suggest a consistent adherence to the described behavior rather than stochastic occurrences. The visualizations in Section III-G showcase this text to driving policy alignment.

### C. Diversity Baselines

**Code Diversity.** To evaluate the diversity of the reward functions, we utilize the Kendall rank correlation coefficient, T, a common statistical metric for assessing the concordance in trends of time series [9], to compare the rank-ordering of rewards. We define the ranked rewards associated with the reward function $\mathcal{R}$ as $\overline{\mathcal{R}}$. The ranking process involves discretizing the reward signals into $K$ distinct ranks with $K$ chosen as an odd positive integer to maintain symmetry. Here, a rank of 1 is assigned to the highest positive reward, a middle rank to zero reward, and a rank of $K$ to the lowest negative reward. Ranks 1 to $\frac{K+1}{2}$ categorize positive rewards, while ranks $\frac{K+1}{2}$ to $K$ categorize negative rewards, both distributed into equal parts. We then mask away the central rank when both reward functions exhibit neutral rewards, focusing our analysis on the agreement and disagreement on active reward states. Our diversity metric does not need to assume continuity and is also scale invariant as it measures similarity according to reward prioritization rather than the absolute reward values. We report the median T results with $p$-value $= 0.05$ at different $K = \{5, 7, 9\}$ values in Table I. We will denote this diversity measure as $\mathcal{D}^C$ .

A high $\mathcal{D}^C$ value, close to 1, indicates similar reward rankings and low diversity. Conversely, a value near 0 suggests moderate diversity due to inconsistent reward correlations. Negative $\mathcal{D}^C$ values, particularly those approaching -1, signify high diversity, as the reward function ranks rewards inversely. In the merge map, a higher similarity is expected due to a common reward that promotes merging. In contrast, other environments displayed lower or negative $\mathcal{D}^C$ values, attributable to fewer simulation constraints.

| Environment | Rank levels ($K$) | Pairwise Kendall Tau ↓ Human Expert | Ours |
|---|---|---|---|
| Intersection | $K = 5$ | -0.0398 | **-0.1352** |
| | $K = 7$ | -0.0645 | **-0.1189** |
| | $K = 9$ | -0.0538 | **-0.1188** |
| Merge | $K = 5$ | **0.2463** | 0.3675 |
| | $K = 7$ | **0.1985** | 0.2252 |
| | $K = 9$ | **0.1970** | 0.3938 |
| Highway | $K = 5$ | 0.0013 | **-0.2525** |
| | $K = 7$ | 0.0306 | **-0.2358** |
| | $K = 9$ | 0.0351 | **-0.2206** |

TABLE I: Code diversity using Kendall Tau correlation

| Methods | Jensen-Shannon Divergence (IQR) ↑ Intersection | Merge | Highway |
|---|---|---|---|
| Random Policy (6 skills) | 0.1197 (0.0019) | 0.2297 (0.0040) | 0.2515 (0.0022) |
| Random Policy (30 skills) | 0.1385 (0.0014) | 0.2250 (0.0084) | 0.3033 (0.0007) |
| Human Expert (5 skills) | 0.1686 (0.0313) | 0.2595 (0.0239) | **0.3686 (0.0442)** |
| DIAYN (6 skills) | 0.0107 (0.0062) | 0.0152 (0.0038) | 0.0254 (0.0021) |
| DIAYN (18 skills) | 0.0163 (0.0039) | 0.0211 (0.0058) | 0.0319 (0.0014) |
| DIAYN (36 skills) | 0.0181 (0.0079) | 0.0083 (0.0027) | 0.0195 (0.0067) |
| Ours (6 skills) | **0.1845 (0.1085)** | **0.3397 (0.0523)** | 0.3039 (0.0729) |

TABLE II: Trajectory diversity using JSD

We benchmarked our results against the default reward functions from the Highway Env simulator that was developed and refined by the community, which we regarded as expert-crafted rewards [16]. We conducted the same experiments on 5 different expert-craft reward functions per environment and reported the median $\mathcal{D}^C$ in Table I. For both the "intersection" and "highway" environment, ***T2D*** consistently yielded lower $\mathcal{D}^C$ values than those of the expert-crafted reward functions, indicating greater diversity in our reward structures. This discrepancy was most notable in the "highway" environment, where ***T2D*** exhibited significantly more negative $\mathcal{D}^C$ values, contrasting that with the positive $\mathcal{D}^C$ values from the expert reward functions. On the "merge" environment, our results indicated less diversity compared to the expert-crafted reward functions, possibly because the expert reward functions did not promote merging behavior as a common reward objective, whereas ours did. Lastly, the histogram in Figure 7a shows that ***T2D*** resulted in a greater spread of code diversity outcomes, particularly by achieving a higher frequency of negative Kendall Tau correlation values. Moreover, it recorded a lower mean pairwise Kendall Tau score than that associated with expert rewards. This highlights our method's ability to generate varied reward structures to promote diverse driving behaviors.

**Driving Policy Diversity.** In this section, we show that ***T2D*** can also generate behaviorally diverse trajectories. We use an existing metric introduced in [17] to measure the trajectory diversity via the Jensen-Shannon Divergence (JSD). We report the median JSD across all agents on 30 different seedings for each map in Table II.

To contextualize these findings, we benchmarked against three different baselines: random behaviors, unsupervised skill acquisition algorithms, and driving policies trained on expert-crafted reward functions [16]. Random behaviors were generated by defining $\pi^P$ as a uniform distribution, where it equally picks an action from the available actions. Then, 30 random behaviors are generated for each map through varied seeding. Next, we compared ***T2D*** to Diversity is

All You Need (DIAYN) [10], an established unsupervised skill acquisition algorithm. We adapted the DIAYN method into a multi-agent setting and trained for 3 different skill counts per map (6, 18, and 36). Our third baseline is against driving policies that were trained on expert-crafted reward functions. We report the median $\mathcal{D}^P$ in Table II. Our results, as summarized in the table, indicate that ***T2D*** surpass random policies and DIAYN-generated policies across all tested scenarios. Notably, ***T2D*** exhibits the highest $\mathcal{D}^P$ in "merge" scenarios, suggesting a greater behavioral diversity compared to other methods. Even in "intersection" and "highway" scenarios, our approach demonstrates competitive diversity, only marginally trailing the human expert in "highway" scenarios. The pairwise JSD values distribution shown in Figure 7b suggests that driving policies derived from explicitly defined reward functions tend to yield a more dispersed and wider spread of policy diversity compared to those from intrinsic reward policies and random behaviors.

## V. CONCLUSION

In our work, we introduce Text-to-Drive (T2D) to generate diverse driving behaviors from natural language descriptions. T2D utilizes an LLM to synthesize behaviors in simulation, constructing a state chart for mapping states to high-level abstractions, thereby enhancing task summarization, policy alignment assessment, and auxiliary reward shaping without human supervision. With our knowledge-driven approach, we demonstrate that T2D generates more diverse trajectories compared to other baselines and offers a natural language interface that allows for incorporating human preference.

## REFERENCES


[1] Alexander Amini, Tsun-Hsuan Wang, Igor Gilitschenski, Wilko Schwarting, Zhijian Liu, Song Han, Sertac Karaman, and Daniela Rus. Vista 2.0: An open, data-driven simulator for multimodal sensing and policy learning for autonomous vehicles. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2419–2426. IEEE, 2022.

[2] Luca Bergamini, Yawei Ye, Oliver Scheel, Long Chen, Chih Hu, Luca Del Pero, Blazej Osinski, Hugo Grimmett, and Peter Ondruska. Simnet: Learning reactive self-driving simulations from real-world observations, 2021.

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[4] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles, 2022.

[5] Yulong Cao, Boris Ivanovic, Chaowei Xiao, and Marco Pavone. Reinforcement learning with human feedback for realistic traffic simulation, 2023.

[6] Wei-Jer Chang, Francesco Pittaluga, Masayoshi Tomizuka, Wei Zhan, and Manmohan Chandraker. Controllable safety-critical closed-loop traffic simulation via guided diffusion, 2023.

[7] Wenhao Ding, Yulong Cao, Ding Zhao, Chaowei Xiao, and Marco Pavone. Realgen: Retrieval augmented generation for controllable traffic scenarios, 2023.

[8] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[9] A. H. El-Shaarawi and Stefan P. Niculescu. On kendall's tau as a test of trend in time series data. *Environmetrics*, 3(4):385–411, 1992.

[10] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function, 2018.

[11] 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.

[12] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. Codebert: A pre-trained model for programming and natural languages, 2020.

[13] 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*, PLDI '19. ACM, June 2019.

[14] Cole Gulino, Justin Fu, Wenjie Luo, George Tucker, Eli Bronstein, Yiren Lu, Jean Harb, Xinlei Pan, Yan Wang, Xiangyu Chen, John D. Co-Reyes, Rishabh Agarwal, Rebecca Roelofs, Yao Lu, Nico Montali, Paul Mougin, Zoey Yang, Brandyn White, Aleksandra Faust, Rowan McAllister, Dragomir Anguelov, and Benjamin Sapp. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research, 2023.

[15] W. Bradley Knox, Alessandro Allievi, Holger Banzhaf, Felix Schmitt, and Peter Stone. Reward (mis)design for autonomous driving, 2022.

[16] Edouard Leurent. An environment for autonomous driving decision-making. https://github.com/eleurent/highway-env, 2018.

[17] Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7204–7213. PMLR, 18–24 Jul 2021.

[18] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models, 2023.

[19] Sivabalan Manivasagam, Shenlong Wang, Kelvin Wong, Wenyuan Zeng, Mikita Sazanovich, Shuhan Tan, Bin Yang, Wei-Chiu Ma, and Raquel Urtasun. Lidarsim: Realistic lidar simulation by leveraging the real world, 2020.

[20] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[21] OpenAI. Generative pre-trained transformer 4 (gpt-4). https://openai.com/research/gpt-4, 2023.

[22] Davis Rempe, Jonah Philion, Leonidas J. Guibas, Sanja Fidler, and Or Litany. Generating useful accident-prone driving scenarios via a learned traffic prior, 2022.

[23] Junha Roh, Chris Paxton, Andrzej Pronobis, Ali Farhadi, and Dieter Fox. Conditional driving from natural language instructions, 2019.

[24] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills, 2020.

[25] Shinya Shiroshita, Shirou Maruyama, Daisuke Nishiyama, Mario Ynocente Castro, Karim Hamzaoui, Guy Rosman, Jonathan DeCastro, Kuan-Hui Lee, and Adrien Gaidon. Behaviorally diverse traffic simulation via reinforcement learning, 2020.

[26] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Sheng Zhao, Shuyang Cheng, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2020.

[27] Simon Suo, Sebastian Regalado, Sergio Casas, and Raquel Urtasun. Trafficsim: Learning to simulate realistic multi-agent behaviors, 2021.

[28] Shuhan Tan, Boris Ivanovic, Xinshuo Weng, Marco Pavone, and Philipp Kraehenbuehl. Language conditioned traffic generation, 2023.

[29] Shuhan Tan, Kelvin Wong, Shenlong Wang, Sivabalan Manivasagam, Mengye Ren, and Raquel Urtasun. Scenegen: Learning to generate realistic traffic scenes, 2021.

[30] Danfei Xu, Yuxiao Chen, Boris Ivanovic, and Marco Pavone. Bits: Bi-level imitation for traffic simulation, 2022.

[31] 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, 2023.