



Titre: Toward Multi-Agent Data-Driven Interactive Autonomous Vehicles
Title:

Auteur: Roger Girgis
Author:

Date: 2026

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Girgis, R. (2026). Toward Multi-Agent Data-Driven Interactive Autonomous Vehicles [Thèse de doctorat, Polytechnique Montréal]. PolyPublie.
Citation: <https://publications.polymtl.ca/76256/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/76256/>
PolyPublie URL:

Directeurs de recherche: Christopher J. Pal
Advisors:

Programme: Génie logiciel
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Toward Multi-Agent Data-Driven Interactive Autonomous Vehicles

ROGER GIRGIS

Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*

Génie informatique

Avril 2026

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Cette thèse intitulée :

Toward Multi-Agent Data-Driven Interactive Autonomous Vehicles

présentée par **Roger GIRGIS**

en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Guillaume-Alexandre BILODEAU, président

Christopher J. PAL, membre et directeur de recherche

Pierre-Yves LAJOIE, membre

Steven L. WASLANDER, membre externe

DEDICATION

To Catti, thank you for your love, patience, and for making life outside this work so full.

To Olivia, may this work inspire you to stay curious and believe in what you can build.

To my parents, thank you for crossing oceans, and for all that entailed.

Tina, I'm finally Dr. Robotss!

ACKNOWLEDGEMENTS

First, I am deeply grateful to my advisor and mentor, Christopher Pal. His wide-ranging expertise in machine learning, his candour, humour, and healthy skepticism toward bureaucracy, together with his confidence in me and his guidance in many aspects of life, gave me exactly the nudge I needed whenever the way forward seemed uncertain. In the end, I believe we were successful at keeping clear of the event horizons of research’s many black holes.

I have been fortunate to collaborate with many outstanding researchers throughout my doctoral studies, both on the projects presented in this thesis and beyond. Our discussions and collaborations have shaped the way I approach research problems, and I am grateful for the ideas, perspective, and generosity they brought to this work. In particular, I would like to thank:

- Felix Heide, for your constant career guidance, your insistence on technical excellence, and for always pushing me to work with greater efficiency.
- Liam Paull, for asking the right questions at the right time, and for your invaluable supervision throughout my studies.
- Martin Weiss, my first collaborator and friend at Mila, with whom I had the joy of sharing many firsts.
- Florian Golemo, Samira El Kahou, and Felipe Codevilla, for your much-needed practical guidance and support at the beginning of my doctoral journey.
- Jim Aldon D’Souza, for your friendship, collaboration, and invaluable contributions to our AutoBots work and beyond. MLVI together strong!
- Julien Roy, for showing me what it means to have a good work-life balance while still producing high-quality work.
- Pierre-Luc Bacon and Joshua Romoff, for the many fruitful discussions about constrained reinforcement learning and multi-objective optimization.
- Alexandre Piché, for your friendship and for introducing me to offline reinforcement learning and exponential tilting.
- Luke Rowe, Anthony Gosselin, and Rodrigue de Schaetzen, for the many deep discussions on the future of autonomous driving, AI, and where it may all be headed. See you on the pickleball courts!

- Bruno Carrez and Azalée Robitaille, for your helpful contributions to different parts of this journey: Bruno, for your work on the simulator for CtRL-Sim; and Azalée, for your help with our constrained GRPO work.

I would also like to express my gratitude to the members of my doctoral supervisory committee, Prof. Guillaume-Alexandre Bilodeau and Prof. Pierre-Yves Lajoie, as well as the external examiner, Prof. Steven Waslander. I sincerely appreciate the fruitful discussions on the ideas described in this thesis and the time spent on the thorough evaluation of my dissertation.

RÉSUMÉ

La recherche présentée dans cette thèse porte sur la modélisation des comportements multi-agents pour les systèmes de conduite autonome. Alors que les travaux en apprentissage automatique se sont longtemps concentrés sur la construction de représentations de scène à partir des données de capteurs (sensor data), nous supposons l'accès à des représentations abstraites fiables de l'environnement. Munis de ces représentations, nous examinons la chaîne de décision des véhicules autonomes (VAs), en particulier les modules de prédiction et de planification, et proposons des méthodes pour les améliorer. Dans la chaîne de planification, le module de prédiction doit produire des prédictions multimodales et précises des agents environnants. Le module de planification doit générer une trajectoire efficace vers la destination tout en respectant des critères de sécurité et de confort. L'évaluation en boucle fermée (closed-loop) de la chaîne de planification est essentielle pour le déploiement des VAs, car elle permet de mesurer la déviation du planificateur entraîné par imitation (imitation drift). Cela exige des simulateurs capables de produire les comportements d'agents "de fond" qui réagissent aux actions du VA.

Nous apportons plusieurs contributions visant une modélisation axée sur les données des interactions entre agents : (1) des Transformers d'ensembles séquentiels produisant des prédictions multimodales et multi-agents sous forme d'ensembles à cardinalité variable, tout en restant permutation-équivalents et en modélisant les interactions temporelles inter-agents; (2) *CtRL-Sim*, un simulateur réactif de comportements appris par renforcement hors ligne, qui remplace le rejeu de journaux par des agents apprenants réagissant à la politique de l'ego et permettant des déploiements stables en boucle fermée, y compris sous attaques adversariales potentielles; (3) des modèles de diffusion latente vectorisée pour synthétiser de nouvelles géométries routières et des états initiaux d'agents dans les scènes de conduite; (4) un cadre d'apprentissage par renforcement contraint pour la spécification des comportements d'agents, qui formule la planification comme une optimisation sous contraintes explicites de sécurité et d'autres exigences comportementales, afin d'arbitrer entre efficacité et respect des contraintes.

Empiriquement, la prévision par ensembles séquentiels offre un moyen efficace de projeter l'évolution future de scènes entières tout en conservant des modes divers et sensibles aux interactions. Le remplacement du rejeu non-réactif par *CtRL-Sim* améliore le réalisme des autres agents dans l'environnement et permet d'évaluer les planificateurs face aux comportements adversariaux. En couplant la génération de scènes initiales avec *CtRL-Sim*, *Scenario*

Dreamer synthétise des simulations réactives entièrement en boucle fermée, ce qui autorise l’étude des effets de l’entraînement et de l’évaluation des planificateurs dans des scénarios nouveaux et de longue durée. Notre approche par renforcement contraint pour la spécification des comportements permet de définir des politiques qui respectent des seuils de sécurité, évitant le besoin de “reward engineering”. Nous étendons ensuite ce cadre de spécification de comportements contraints à Group Relative Policy Optimization (GRPO), en introduisant *Constrained GRPO*, et en établissant une approche simple pour assurer une application fiable des contraintes dans une optimisation de politique sans critique. Pris ensemble, ces composants constituent les bases d’une pile comportementale où prévision, planification et simulation sont co-conçues pour le développement en boucle fermée des VAs. Nous espérons que ce travail servira de fondation à de futures recherches en modélisation des comportements pour la conduite autonome.

ABSTRACT

The research presented in this thesis focuses on multi-agent behaviour modelling for autonomous driving systems. While prior machine learning research largely concentrated on building scene representations from sensor data, we assume access to reliable abstract world representations. With this assumption, we study current approaches through the decision-making stack of autonomous vehicles (AVs), namely the prediction and planning modules, and propose approaches for improving them. The prediction module of an AV should produce accurate, multi-modal forecasts of surrounding agents, while the planning module should generate an efficient route toward its destination while satisfying safety and comfort standards. Closed-loop evaluation of the planning module is essential to the deployment of the decision-making stack as it can measure imitation drift. This requires simulation frameworks that can produce the behaviours of background agents which react to the AV’s actions.

We make several contributions on these components, with the goal of modelling the interactive behaviours of agents in a data-driven manner: (1) Sequential Set Transformers that produce multi-modal, multi-agent forecasts as variable-cardinality sets, remaining permutation-equivariant while modelling inter-agent interactions over time; (2) CtRL-Sim, a reactive behaviour simulator trained with offline reinforcement learning, which replaces log-replay with learned agents that respond to the ego policy and support stable closed-loop rollouts with potential adversarial attacks; (3) Vectorized latent diffusion models for synthesizing novel road geometries and initial agent states in driving scenes; and (4) Constrained reinforcement learning for behaviour specification that frames planning as optimization under explicit safety and other behavioural constraints, trading off efficiency with constraint satisfaction.

Empirically, sequential set forecasting provides an efficient way to forecast the future rollout of entire scenes while maintaining diverse, interaction-aware modalities. Replacing non-reactive log-replay simulation with CtRL-Sim improves the realism of surrounding agents and allows AV practitioners to evaluate their planners against long-tail behaviours. Coupling the generative model of initial driving scenes with CtRL-Sim, Scenario Dreamer synthesizes entire closed-loop reactive simulations, affording the ability to study the effects of training and evaluating planners in novel and long scenarios. Our constrained RL approach to behaviour specification allows us to define policies that respect safety thresholds, circumventing the need for reward engineering. We further extend this constrained behaviour specification framework to Group Relative Policy Optimization (GRPO), introducing *Constrained GRPO* and establishing a simple recipe for reliable constraint enforcement in critic-free policy optimiza-

tion. Together, these components form the basis of a behaviour stack in which forecasting, planning, and simulation are co-designed for closed-loop development of AVs. We hope that the work presented in this thesis serves as the basis for future research in the field of behaviour modelling for AVs.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	vi
ABSTRACT	viii
LIST OF TABLES	xiii
LIST OF FIGURES	xiv
LIST OF SYMBOLS AND ACRONYMS	xvi
LIST OF APPENDICES	xix
CHAPTER 1 INTRODUCTION	1
1.1 Multi-Agent Motion Forecasting with Sequential Set Transformers	4
1.2 Controllable Multi-Agent Simulation via Offline RL	4
1.3 Vectorized Diffusion Models for Scenario Generation	5
1.4 Constraint-Satisfying Policies in Reinforcement Learning	6
1.5 Thesis Organization	7
CHAPTER 2 BACKGROUND	9
2.1 Attention Mechanisms	9
2.1.1 Soft Attention	9
2.1.2 Transformer and Set-Attention Building Blocks	10
2.2 Latent Variable Models	12
2.2.1 Variational Inference and VAEs	14
2.2.2 Diffusion Models	14
2.3 Reinforcement Learning	18
2.3.1 Value Functions	18
2.3.2 Policy Gradient Methods	20
CHAPTER 3 LITERATURE REVIEW	26
3.1 Data-Driven Simulation for Autonomous Driving	26
3.1.1 Simulation Environments for Autonomous Driving	26

3.1.2	Initial Scene Generation: Modelling $P(\mathcal{M}, \mathbb{X}_0 \mid c)$	27
3.1.3	Agent Behaviour Simulation: Modelling $P(\mathbb{X}_{1:T} \mid \mathcal{M}, \mathbb{X}_0, c)$	28
3.2	Behaviour Specification Through Constrained RL	30
3.2.1	Constrained Reinforcement Learning	30
3.2.2	Reward Specification	31
3.2.3	Reinforcement Learning in Video Games	31
3.2.4	Constrained RL for LLMs	32
CHAPTER 4 ARTICLE 1: LATENT VARIABLE SEQUENTIAL SET TRANS-		
	FORMERS FOR JOINT MULTI-AGENT MOTION PREDICTION	34
4.1	Abstract	34
4.2	Introduction	35
4.3	Background	36
4.4	Latent Variable Sequential Set Transformers	37
4.4.1	Encoder: Input Sequence Representation	37
4.4.2	Decoder: Multimodal Sequence Generation	39
4.4.3	Training Objective	40
4.5	Experiments	41
4.5.1	NuScenes, A Real-World Driving Dataset	42
4.5.2	Argoverse Results	43
4.5.3	TrajNet++ Synthetic Dataset	44
4.5.4	Omniglot, Stroke-based Image Generation	46
4.6	Related Work	46
4.7	Conclusion	48
CHAPTER 5 CONTROLLABLE DRIVING BEHAVIOUR SIMULATION WITH		
	OFFLINE REINFORCEMENT LEARNING	50
5.1	Introduction	51
5.2	CtRL-Sim	52
5.2.1	Controllable Simulation via Offline RL	53
5.2.2	Multi-Agent Behaviour Simulation Architecture	55
5.3	Experiments	58
5.3.1	Experimental Setup	58
5.3.2	Results	60
5.4	Conclusion	64

CHAPTER 6	SYNTHESIZING INITIAL SCENES FOR FULLY DATA-DRIVEN	
	SIMULATION	65
6.1	Introduction	66
6.2	Proposed Framework	68
6.2.1	Vectorized Latent Diffusion for Initial Scene Generation	68
6.2.2	Full Scenario Simulation Framework	72
6.3	Experiments	74
6.3.1	Experimental Setup	74
6.3.2	Results	76
6.4	Conclusion	78
CHAPTER 7	SPECIFYING DESIRABLE BEHAVIOURS WITH CONSTRAINED	
	REINFORCEMENT LEARNING	80
7.1	Introduction	81
7.2	Empirical Analysis of Reward Engineering	82
7.3	Constrained Markov Decision Processes	86
7.4	Direct Behaviour Specification	87
7.4.1	Indicator cost functions	87
7.4.2	Multiplier Normalization	89
7.4.3	Constrained GRPO	90
7.5	Experiments	92
7.5.1	Constrained GRPO: Gridworld Environment	93
7.5.2	Constrained GRPO: Autonomous Driving Benchmark	95
7.6	Conclusion	100
CHAPTER 8	CONCLUSION	103
8.1	Synthèse des travaux / Summary of Works	103
8.2	Limitations de la solution proposée / Limitations	104
8.3	Context of the Work and Author Contributions	105
8.3.1	AutoBots	105
8.3.2	CtRL-Sim	106
8.3.3	Scenario Dreamer	107
8.3.4	Constrained RL	108
8.4	Améliorations futures / Future Research	109
APPENDICES		139

LIST OF TABLES

Table 3.1	Offline policy modelling approaches in prior work	29
Table 4.1	Quantitative Results on the nuScenes dataset	43
Table 4.2	Quantitative Results on the Argoverse test set	45
Table 4.3	TrajNet++ ablation studies for a multi-agent forecasting scenario . .	45
Table 5.1	Multi-agent simulation results over 1000 test scenes.	61
Table 5.2	Adversarial scenario generation results over 1000 test scenes.	62
Table 6.1	Quantitative Assessment of Lane Graphs	76
Table 6.2	Quantitative Assessment of Agent Placement	77
Table 6.3	Ablation of Vectorized Latent Diffusion	77
Table 6.4	RL Planner Results	78
Table 7.1	Stage 1 NAVSIM-v2 Results	99
Table 7.2	Stage 2 NAVSIM-v2 Results	100
Table A.1	Hyperparameters used for AutoBots across all four datasets.	144
Table A.2	NuScenes Ablation Study.	148
Table A.3	NuScenes Quantitative Results by Entropy Regularization	149
Table A.4	TrajNet++ ablation studies for a multi-agent forecasting scenario. . .	150
Table A.5	TrajNet ablation studies for a multi-agent forecasting scenario. . . .	150
Table B.1	Breakdown of Meta-JSD results	162
Table B.2	Ablations of CTG++ on Multi-agent simulation results over 1000 test scenes	163
Table B.3	Multi-agent simulation results over 1000 test scenes with action tem- perature = 1.5 over 3 seeds	167
Table B.4	CtRL-Sim-FT’s tilting Results	167
Table C.1	Simulation Methods Comparison	170
Table D.1	Hyper-Parameters for Arena Experiments	194
Table D.2	Hyper-Parameters for OpenWorld Experiments	198

LIST OF FIGURES

Figure 1.1	A typical modular decision-making stack for autonomous driving. . .	3
Figure 2.1	The Agent-Environment Interface in Reinforcement Learning	19
Figure 4.1	AutoBots Overview	37
Figure 4.2	AutoBots Architecture Details	38
Figure 4.3	NuScenes Trajectory Prediction	44
Figure 4.4	Omniglot qualitative results	47
Figure 5.1	CtRL-Sim Overview	52
Figure 5.2	CtRL-Sim Architecture	56
Figure 5.3	Qualitative results of multi-agent simulation with CtRL-Sim.	57
Figure 5.4	Effects of exponential tilting.	62
Figure 5.5	Qualitative results of vehicle-vehicle and vehicle-edge tilting.	63
Figure 6.1	Overview of initial scene generation with our vectorized latent diffusion model	69
Figure 6.2	Qualitative results of Vectorized Environment Generation	70
Figure 6.3	Example of Scenario Dreamer Inpainting	73
Figure 7.1	Overview of Environments	82
Figure 7.2	Driving Environment Overview	83
Figure 7.3	Exposing reward engineering with one and two constraints	84
Figure 7.4	Exposing reward engineering with three constraints	85
Figure 7.5	Fixed-weights GRPO comparing Scalarized Rewards with Scalarized Advantages	94
Figure 7.6	Learning Dynamics of Constrained GRPO with $\tilde{d}_{\text{lava}} = \tilde{d}_{\text{battery}} = 0.01$	95
Figure 7.7	Multiple Constraints on NAVSIM training	101
Figure A.1	AutoBots Context Encoder	143
Figure A.2	TrajNet Map Example.	144
Figure A.3	AutoBots’s Inference Rate	147
Figure A.4	Influence of entropy loss on particle experiment.	149
Figure A.5	TrajNet++ qualitative results.	151
Figure A.6	TrajNet qualitative results.	152
Figure A.7	TrajNet qualitative results 1/4.	153
Figure A.8	TrajNet qualitative results 2/4.	154
Figure A.9	TrajNet qualitative results 3/4.	154
Figure A.10	TrajNet qualitative results 4/4.	155

Figure A.11	Omniglot Task 1 Additional Results.	156
Figure A.12	Omniglot Task 1 Failure Cases.	157
Figure A.13	Omniglot Task 2 Additional Results.	158
Figure B.1	Qualitative results of the effects of positive tilting	165
Figure B.2	Qualitative results of vehicle-vehicle tilting	166
Figure B.3	Effects of exponential tilting	168
Figure B.4	Failure cases.	169
Figure C.1	Lane-conditioned object generation and initial scene generation qualitative results	172
Figure C.2	Scenario Dreamer inpainting	173
Figure C.3	Qualitative results comparison across methods (set 1)	174
Figure C.4	Qualitative results comparison across methods (set 2)	175
Figure C.5	Scene Diversity	176
Figure C.6	Behaviour Simulation Examples	176
Figure C.7	Partitioned vs. Non-partitioned Scenes	183
Figure C.8	Proposed ordering scheme	185
Figure C.9	Visualization of the k -disks tokenization scheme	186
Figure D.1	Effects of Multiplier Normalization in the Arena Environment	195
Figure D.2	Single Behaviour Specification in the Arena Environment	195
Figure D.3	Multiple Behaviour Specification in the Arena Environment	197
Figure D.4	Multiple Behaviour Specification in the OpenWorld Environment	199
Figure D.5	Learning Dynamics of Constrained GRPO with $\tilde{d}_{\text{lava}} = 0.01$	202
Figure D.6	Learning Dynamics of Constrained GRPO with $\tilde{d}_{\text{lava}} = 0.01$ and $\tilde{d}_{\text{battery}} = 0.1$	203

LIST OF SYMBOLS AND ACRONYMS

AD	Autonomous Driving
ADE	Average Displacement Error
AV	Autonomous Vehicle
BEV	Bird’s-Eye View
Box2D	2D Rigid-Body Physics Engine
CMDP	Constrained Markov Decision Process
DAC	Drivable Area Compliance
DDC	Driving Direction Compliance
DDPM	Denoising Diffusion Probabilistic Model
DiT	Diffusion Transformer
DT	Decision Transformer
ELBO	Evidence Lower Bound
EM	Expectation–Maximization
EPDMS	Extended PDM Score
FD	Fréchet Distance
FDE	Final Displacement Error
FFN	Feed-Forward Network
FLOPs	Floating-Point Operations
FOV	Field of View
FT	Fine-Tuned
GAIL	Generative Adversarial Imitation Learning
GAE	Generalized Advantage Estimation
GAN	Generative Adversarial Network
GDA	Gradient Descent–Ascent
GELU	Gaussian Error Linear Unit
GRPO	Group Relative Policy Optimization
GT	Ground Truth
GPU	Graphics Processing Unit
GPUh	GPU-hours
HD	High-Definition
IDM	Intelligent Driver Model
IL	Imitation Learning
JSD	Jensen–Shannon Divergence

KL	Kullback–Leibler Divergence
LiDAR	Light Detection and Ranging
LLM	Large Language Model
LN	Layer Normalization
LQR	Linear–Quadratic Regulator
LSTM	Long Short-Term Memory
MAB	Multi-Head Attention Block
MABD	Multi-Head Attention Decoder Block
MB-RCP	Model-Based Reward-Conditioned Policies
MDP	Markov Decision Process
ME	Mode Entropy
MHAB	Multi-Head Attention Block
MHADB	Multi-Head Attention Decoder Block
MHA	Multi-Head Attention
MHSA	Multi-Head Self-Attention
MLP	Multi-Layer Perceptron
MOBIL	Minimizing Overall Braking Induced by Lane changes
MuJoCo	Multi-Joint dynamics with Contact
MTGS	Multi-Traversal Gaussian Splatting
NC	No At-Fault Collisions
NPC	Non-Player Character
PDM	Predictive Driver Model
PE	Positional Encoding
PPO	Proximal Policy Optimization
RBC	Reweighted Behavior Cloning
RCP	Reward-Conditioned Policy
RFT	Reinforcement Learning Fine-Tuning
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RWR	Reward Weighted Regression
SAC	Soft Actor–Critic
SFT	Supervised Fine-Tuning
TD	Temporal Difference
TLC	Traffic Light Compliance
TRPO	Trust Region Policy Optimization

VAE	Variational Autoencoder
VLM	Vision–Language Model
WAT	Weighted Average Terms
WOMD	Waymo Open Motion Dataset
WOSAC	Waymo Open Sim Agents Challenge

LIST OF APPENDICES

Appendix A Latent Variable Sequential Set Transformers & AutoBots 139

Appendix B Controllable Behaviour Simulation & CtRL-Sim 159

Appendix C Vectorized Latent Diffusion & Scenario Dreamer 170

Appendix D Direct Behaviour Specification & Constrained GRPO 189

CHAPTER 1 INTRODUCTION

Autonomous driving (AD) is an interdisciplinary engineering grand challenge at the intersection of robotics, computer vision, and artificial intelligence. The goal is to enable vehicles to navigate safely and efficiently without human intervention. Solving the AD problem has significant implications for transportation safety, efficiency, and accessibility. In a classical AD system, the software stack is built in a modular manner, with distinct components for perception, prediction, planning, and control. Perception modules process sensor inputs to build a representation of the environment (e.g. detecting other vehicles, pedestrians, and the road layout). The prediction module generates forecasts of the future trajectories of other dynamic agents (vehicles, pedestrians, etc.), while the planning module uses this information to decide the ego vehicle’s future path or maneuver. Finally, the control module executes the low-level commands to follow the planned trajectory as faithfully as possible, e.g., steering, throttle, braking.

Most of the early work in machine learning for autonomous driving was concentrated in perception, with a focus on improving the accuracy of object detection and scene understanding, whereas planning and decision-making often relied on rule-based or optimization-based methods. Notably, deep neural networks have revolutionized perception tasks in autonomous driving, such as detecting and classifying objects [1, 2] or interpreting camera and LiDAR data [3, 4] in harsh weather conditions. To support machine learning research in these problems, many open-source datasets and benchmarks have been proposed [5, 6, 7, 8].

The modular nature of the autonomy stack plays a crucial role in allowing for explainability and safety of generated plans by keeping planners as human-engineered frameworks. However, we argue that this limits our ability to leverage large driving datasets to improve decision-making. Very recently, we have witnessed a surge in learning-based approaches on the problems of multi-agent motion forecasting and ego-agent trajectory planning and decision making. Both academia and industry have increasingly found it important to incorporate learning-based methods for prediction and planning, integrating them with or even replacing classical rule-based algorithms.

We assume we are provided with good abstract representations from the perception stack, and are tasked with modeling the distribution of possible multi-agent future worlds. Following this, we are then tasked with generating a safe and efficient plan along a predefined route, taking into account the multi-agent context. The first of these problems, namely multi-agent motion forecasting, demands the ability to model the (latent) intentions of other agents and

the ability to forecast the joint probability distribution over their possible paths. While recurrent neural networks [9, 10] can model the sequential nature of agent trajectories, a good generative model should also model the multi-agent social interaction nature of the problem. Early deep learning approaches tackled this problem with pooling mechanisms for pedestrian forecasting [11], and in autonomous driving specifically [12, 13]. More recently, the attention mechanism emerged as a good alternative for modelling the social interaction between agents [14, 15, 16, 17, 18].

The problem of generating local plans and actions for the ego-agent comes with its own set of challenges. The autonomous agent is required to meet certain safety standards while ensuring that the plans it generates are feasible, efficient, and comfortable for the riders. Some recent modular stacks join the prediction and planning modules into one fully differentiable stack [19, 20]. This allows an agent to model what is happening around it and plan a route using a classical planning module, while employing differentiable cost functions to effectively update the plan. Reinforcement learning (RL) has emerged as a promising approach for solving the decision-making problem, where an agent must learn to take optimal actions through trial-and-error. This has the important side-effect of allowing the agent to learn closed-loop behaviours, mitigating imitation drift issues from imitation-based policies at inference [21, 22]. The success of RL in games [23] has inspired its application to driving for tasks like route planning, lane changing, and collision avoidance. However, an important limitation of RL for driving is the behavior specification bottleneck: many objectives (safety, legality, comfort, progress, social compliance) must be encoded into a single scalar reward, and different scalarizations can induce qualitatively different behaviors. As a result, it can be difficult to exert fine-grained control over the learned policy, whereas classical planners can expose these objectives and constraints explicitly [24, 25].

An important shortcoming of RL is its reliance on a simulator to learn closed-loop optimal policies. In driving environments, the simulator needs access to the rich distribution of possible agent behaviours in order for the RL planner to learn how its actions affect its environment. The rise of deep generative models has opened the door for data-driven simulators that can support driving requirements. The appeal of data-driven approaches lies in their ability to generate more realistic agent behaviours compared to heuristic-based simulators [26, 27, 28]. Driven by this desideratum, Waymo released the Waymo Open Sim Agents Challenge (WOSAC) [29] in which participants must model the multimodal autoregressive future rollout of up to 8 agents in driving scenes. Successful approaches are those that assign high likelihood to the log-replay data. However, an issue with optimizing this imitation objective is that one recovers the log-replay data distribution. Since datasets are usually composed of near-optimal driving, planning policies trained in these environments will not learn how to

deal with long-tail events (e.g., dangerous drivers).

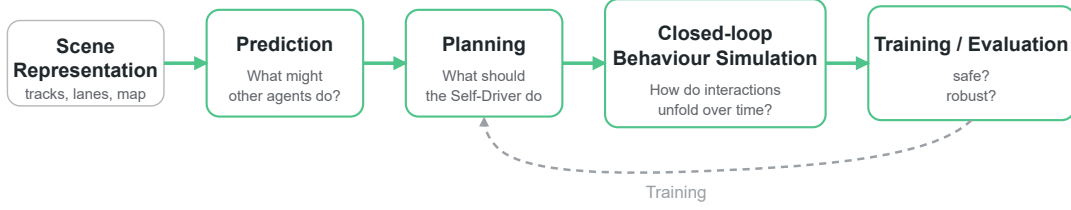


Figure 1.1 **A typical modular decision-making stack for autonomous driving.** Scene representations are used for prediction and planning, while closed-loop behaviour simulation supports training and evaluation of the planning stack.

These limitations reveal a broader theme: robust autonomous driving depends on understanding, predicting, controlling, and simulating the behaviour of agents in interactive driving scenes. This motivates the central thesis of this dissertation:

Robust autonomous driving systems require data-driven behaviour modelling.

Objectives. The objectives of this thesis are to develop data-driven methods for modelling, predicting, and controlling agent behaviours in autonomous driving systems. Concretely, we aim to:

1. Design generative models that capture the joint, multimodal behaviour of interacting agents.
2. Develop controllable and reactive simulation frameworks for closed-loop evaluation and training.
3. Formulate principled reinforcement learning approaches for planning that enable explicit behaviour specification under safety and performance constraints.

Figure 1.1 summarizes how this thesis fits within a typical modular decision-making stack. Starting from scene representations, the stack must predict how other agents may behave, plan what the ego vehicle should do, and evaluate and/or update these decisions in closed-loop simulation. In the following sections, we highlight the specific solutions proposed in this thesis for prediction, planning, and closed-loop behaviour simulation.

1.1 Multi-Agent Motion Forecasting with Sequential Set Transformers

A fundamental challenge in autonomous driving is motion forecasting: the problem of predicting how other agents (e.g., vehicles, cyclists, pedestrians) will behave in the future. This is inherently a multi-agent problem – an autonomous vehicle (AV) must reason about the *joint* behaviour of many interacting agents, each influencing other surrounding agents and each having their own latent intention. In Chapter 4, we present our proposed solution: AutoBots, a latent variable sequential set transformer architecture. This is a chapter by article and the corresponding article is:

R. Girgis, F. Golemo, F. Codevilla, M. Weiss, J. A. D’Souza, S. E. Kahou, F. Heide, and C. J. Pal, “Latent Variable Sequential Set Transformers for Joint Multi-Agent Motion prediction,” in ICLR, 2022

In this work, we model the problem of multi-agent motion forecasting as a time-evolving set of elements; at each timestep we have a set of agents where each agent has associated state attributes (e.g., position, velocity, agent type). At its core, Autobots is an encoder-decoder Transformer [30] architecture. The encoder first generates an embedded representation of the social-temporal history of all agents using multi-head self-attention layers. The decoder then employs learnable seed parameters spanning the future horizon, and applies a series of transformer decoder blocks with attention to the encoder-generated context tensors. In order to model the latent intention of each agent, we introduce a discrete latent variable, which allows us to compute exact likelihoods [31]. Crucially, the discrete latent variables are represented using seed query parameters spanning the number of possible modes being modeled and the number of future timesteps. This design makes AutoBots both efficient and effective at modelling the joint multi-modal distribution over future multi-agent trajectories.

AutoBots was well received in the literature and industry, having been cited over 280 times since its publication, with 117 citations last year alone. It was an important work because it demonstrated the utility of permutation-equivariant factorized attention blocks for improving multi-agent motion prediction for autonomous driving, by effectively capturing agent interactions and producing rich, diverse predictions that can be used by a planning algorithm.

1.2 Controllable Multi-Agent Simulation via Offline RL

An important aspect of the deployment of any autonomy stack is rigorous testing. Traditional evaluation relies on replaying logged real-world driving data, with the behaviour of other agents being replayed. This has the obvious drawback that agents cannot react to the

actions of the planner. Heuristically-driven policies have been proposed to alleviate the reactivity problem, with hard-coded rules for how agents should behave in the vicinity of other agents[32, 33]. However, such policies tend to produce unrealistic behaviours which can provide an incorrect signal of the robustness of the planner being evaluated. Moreover, rule-based approaches can produce behaviours which lack diversity, further limiting their applicability to measuring robustness. There has been a recent surge in research for learning how to simulate realistic and diverse behaviours from real-world data [27, 34, 35, 36, 37]. While these approaches provide more realistic, reactive and diverse behaviours, either they are not easily controllable, or require iterative sampling procedures to produce interesting behaviours. In Chapter 5, we propose CtRL-Sim [26], a framework for controllable behaviour simulation via offline reinforcement learning. The contents of this chapter are a re-written version of our publication at the Conference on Robot Learning in 2024,

*L. Rowe, *R. Girgis, A. Gosselin, B. Carrez, F. Golemo, F. Heide, L. Paull, and C. Pal, “CtRL-sim: Reactive and controllable driving agents with offline reinforcement learning,” in CoRL, 2024. (* joint co-first author, order determined by a coin flip).

In this work, we first build on the Nocturne simulator [38] by adding the Box2d physics engine, which ensures the agent motion through scenes remains realistic. We then replay the logged scenarios from the Waymo Open Motion Dataset [39] through the physics-enhanced Nocturne environment recording actions, rewards, and states of all agents. At the heart of this framework is its offline reinforcement learning formulation, where we train a next-token prediction transformer to jointly model the actions, rewards and state rollouts of all agents. At test time, we can produce nominal reactive driving behaviours for all agents. To enable inference-time controllability, we employ the exponential tilting mechanism [40, 41], enabled by the choice of decomposition of the joint distribution. We present experiments showcasing the realism of the produced behaviours compared to prior approaches, and the effectiveness of the exponential tilting control knobs. Since its publication, it has been well received in the literature and industry, having been cited a total of 20 times.

1.3 Vectorized Diffusion Models for Scenario Generation

In the previous section, we highlighted our proposed solution for controllable and reactive simulation using the CtRL-Sim framework. Agents are randomly initialized in predefined road maps, and learn to navigate towards a goal while minimizing collisions and off-road rates through an aggregated reward function. Cornelisse et al. [42] showed that generalization performance of learned agents on unseen road structures scales with the number of maps used

during RL training. While there are large datasets of recorded driving logs and corresponding maps, current simulators [28, 38, 43] only cover 1750km from the Waymo Open Motion Dataset [39] while a human driver averages 22,000 km annually. In Chapter 6, we present our novel vectorized latent diffusion model [44], to generate novel road geometries along with initial agent bounding boxes in abstract representation spaces. The contents of this chapter are a re-written version of our publication at the Conference on Computer Vision and Pattern Recognition in 2025,

L. Rowe, R. Girgis, A. Gosselin, L. Paull, C. Pal, and F. Heide, “Scenario dreamer: Vectorized latent diffusion for generating driving simulation environments,” CVPR, pp. 17 207–17 218, 2025 (© 2025 IEEE)

Our proposed model can generate large map structures thanks to its inpainting capabilities. Combining the Vectorized latent diffusion model with our CtRL-Sim framework (Chapter 5), we propose Scenario Dreamer, a framework for generating simulation environments. Evaluating pretrained RL agents in Scenario Dreamer-generated scenes exposes significant behavioural shortcomings, mainly higher collision rates and lower success rates, with the effect amplified on long routes. Although it has only recently been published, this work has been well received in the literature, having been cited 18 times. Its core novelty is its vectorized latent diffusion model which provides a good prior on abstract representations and allows for efficient generation of scenes.

1.4 Constraint-Satisfying Policies in Reinforcement Learning

Reinforcement learning (RL) has emerged as a powerful framework for decision-making in Markov Decision Processes. Classically, RL aims to maximize the sum of scalar rewards, called the return, by interacting with an environment through trial and error. In the context of driving however, one needs to maximize efficiency towards reaching a predefined goal while minimizing violations of certain behavioural constraints. These can include safety constraints such as reducing collisions to an extremely low percentage, reducing the rate agents veer off road, or even satisfying realism constraints. In Chapter 7, we present a framework that builds on the Constrained Markov Decision Processes (CMDPs) to restrict the feasible policy space of a reward-maximizing agent to ones that adhere to behavioural constraints [25]. Part of the work presented in Chapter 7 was published at the International Conference on Machine Learning in 2022,

J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. J. Pal, “Direct behavior specification via constrained reinforcement learning,” in ICML, 2022

In this work, we propose a direct behaviour specification framework which lets designers specify the desired behaviour rates of agents in an intuitive way. We first show the effectiveness of this framework in the world of video-games where agents must navigate through complex environments toward their predefined goals while respecting certain constraints (e.g., not jumping too often). This published framework was well received in the literature having been cited over 50 times since its publication. Its main contribution is that designers can specify desirable behaviour rates using the constrained reinforcement learning framework, avoiding expensive reward engineering.

More recently, we extend this framework to Group Relative Policy Optimization (GRPO) algorithm for RL finetuning (RFT) of large vision-language models. This work is currently under review and is available as an arXiv preprint:

*R. Girgis, *R. de Schaetzen, L. Rowe, A. Robitaille, C. Pal, L. Paull. "Constrained Group Relative Policy Optimization." Under review, 2026. (* joint co-first author).

Specifically, we mathematically analyze the effects of combining multiple reward components with GRPO, and formulate a principled way to reliably improve the advantage estimation in the constrained RL framework. We show the scalability of combining the behaviour specification framework with our improved advantage estimation for GRPO for the NAVSIM end-to-end pseudo-simulation driving challenge. Overall, we show that this framework can play an important role in the decision-making framework of autonomous vehicles, providing designers with an intuitive way to specify desirable behaviours. The content of Chapter 7 is a revised version of the original publication, and further incorporates our Constrained GRPO method and experiments described above.

1.5 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 reviews the background needed for the thesis, including attention mechanisms, latent variable models, diffusion models, and reinforcement learning. Chapter 3 reviews prior literature relevant to the proposed approaches in this thesis. We then present our contributions in turn. Chapter 4 details *AutoBots*, a latent-variable sequential set transformer for forecasting joint, multi-modal, multi-agent futures. In Chapter 5, we propose a framework for synthesizing long-tail, closed-loop behaviour simulations using offline reinforcement learning. Chapter 6 develops a vectorized latent diffusion model for generating new road-map geometries. Combined with the framework from Chapter 5, this enables fully data-driven world simulation with adversarial agent behaviours to train or evaluate driving policies. Chapter 7 introduces our direct behaviour-specification

framework based on constrained reinforcement learning, and extends it to modern critic-free policy optimization frameworks to enable finetuning of large vision-language models.

The work presented in this thesis demonstrates how modern machine learning, from Transformers to diffusion models and constrained RL, can be employed to address the unique challenges of driving in multi-agent, highly dynamic environments with strict safety requirements. Collectively, we believe our work moves the field closer to autonomous vehicles that can predict, plan, and interact with the complexity of the real world with greater reliability and safety awareness.

CHAPTER 2 BACKGROUND

In this Chapter, we present the key theoretical concepts and frameworks we build our methods upon. We start by presenting the Transformer architecture along with its key architectural components. We employ multi-head attention in Chapters 4, 5 and 6 for dealing with spatial and temporal data structures. Next, we turn our attention to latent variable models, a key concept for dealing with multimodality in the AutoBots work of Chapter 4, while variational autoencoders and diffusion models were the basis for the vectorized latent diffusion model in Chapter 6. The final major framework we cover is reinforcement learning. We focus on the general framework and present key algorithms we build upon in this thesis, particularly in Chapters 7, 5 and 6.

2.1 Attention Mechanisms

In this thesis, we make extensive usage of the attention mechanism. It is particularly attractive for attending to other road users, an essential part of making driving decisions. We employ the attention mechanisms in our models presented in the next Chapters. In this section, we explore some of the attention mechanisms that have been proposed in the past.

2.1.1 Soft Attention

One of the big successes of attention mechanisms is in neural machine translation. Earlier approaches [45, 46] used an encoder-decoder framework where the encoder is used to encode the input sentence into a compressed representation and a decoder was used to generate the new sentence. However, this compressed representation behaves as a bottleneck, with it being tasked with representing all the information from the input sequence.

Bahdanau et al. [47] recognized this problem and proposed an attention mechanism to circumvent it. In their seminal work, instead of using the bottleneck features produced by the encoding RNN at every timestep, they allow the decoder to attend to all hidden features produced by the encoder, effectively producing a context c_i for every decoded word y_i . The output sequence is modeled by $s_i = f_\phi(s_{i-1}, y_{i-1}, c_i)$ where the s_i is the hidden state of the decoder RNN. The attention mechanism used to produce the context c_i is as follows,

$$c_i = \sum_{j=1}^t \alpha_{ij} h_j \quad (2.1)$$

where $h_{1:t}$ is the set of all hidden representations produced by the encoder, and α_{ij} are the attention weights produced by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^t \exp(e_{ik})} \quad (2.2)$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model that measures the similarity between the previous hidden state of the decoder and the j th encoder hidden state.

2.1.2 Transformer and Set-Attention Building Blocks

We review the core components of the Transformer architecture introduced by [48] and the Set Transformer of Lee et al. [49], adopting notation consistent with those works.

Scaled Dot-Product Attention. Given a set of n_q query vectors $\mathbf{Q} \in \mathbb{R}^{n_q \times d_k}$, and n_k key-value pairs $\mathbf{K} \in \mathbb{R}^{n_k \times d_k}$ and $\mathbf{V} \in \mathbb{R}^{n_k \times d_v}$, a single (scaled) attention head maps queries to outputs by

$$\text{Attn}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right) \mathbf{V}, \quad (2.3)$$

where d_k is the key/query dimension and d_v is the value dimension. We bring the reader's attention to the similarity between Equations 2.1 and 2.2 to Equation 2.3: the main difference being that alignment model used to generate the softmax logits is generated by a scaled dot-product between the keys and the query vectors.

Multi-Head Attention (MHA). Multi-head attention applies h parallel projections of queries, keys, and values, computes attention of Equation 2.3 in each projected space. The output representation is then concatenated and linearly projected back to the hidden dimension of the model:

$$\begin{aligned} \text{MHA}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{concat}(\text{head}_1, \dots, \text{head}_h) \mathbf{W}^O, \\ \text{head}_i &= \text{Attn}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \end{aligned} \quad (2.4)$$

where the learnable parameters are $\mathbf{W}_i^Q \in \mathbb{R}^{d_k \times d_k^{(h)}}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_k \times d_k^{(h)}}$, $\mathbf{W}_i^V \in \mathbb{R}^{d_v \times d_v^{(h)}}$ for $i = 1, \dots, h$, and $\mathbf{W}^O \in \mathbb{R}^{(h d_v^{(h)}) \times d_{\text{out}}}$. Typically $d_k^{(h)} = d_k/h$ and $d_v^{(h)} = d_v/h$.

Self-Attention on Sets. A useful special case is *self-attention*, where the same input provides queries, keys, and values. For an input set (or sequence) $\mathbf{X} \in \mathbb{R}^{n_x \times d}$,

$$\text{MHSA}(\mathbf{X}) \equiv \text{MHA}(\mathbf{X}, \mathbf{X}, \mathbf{X}). \quad (2.5)$$

Self-attention enables all elements of the set to interact and is a central mechanism in Set Transformer layers [49]. In the context of autonomous driving where multiple agents are interacting, we can view these agent states as a set per timestep. The *MHSA* block is therefore quite useful for this problem.

Feed-Forward Network (rFFN). Following Vaswani et al. [48], we use a row-wise feed-forward network, $\text{rFFN}(\mathbf{H}) = \phi(\mathbf{H}\mathbf{W}_1 + \mathbf{b}_1)\mathbf{W}_2 + \mathbf{b}_2$, applied independently to each row, with a nonlinearity ϕ (e.g., ReLU or GELU). This nonlinearity processes the output token representations, transforming them following the rich attention to other set/sequence elements.

Multi-Head Attention Block (MHAB). We adopt a residual + LayerNorm (LN, [50]) design with an attention sublayer followed by an rFFN sublayer. Given an input $\mathbf{X} \in \mathbb{R}^{n_x \times d}$ and a conditioning context $\mathbf{C} \in \mathbb{R}^{n_c \times d}$, a (cross-)attentional MHAB computes

$$\begin{aligned}\mathbf{H} &= \text{LN}(\mathbf{X} + \text{MHA}(\mathbf{X}, \mathbf{C}, \mathbf{C})), \\ \text{MHAB}(\mathbf{X}, \mathbf{C}) &= \text{LN}(\mathbf{H} + \text{rFFN}(\mathbf{H})).\end{aligned}\tag{2.6}$$

When $\mathbf{C} = \mathbf{X}$, Eq. (2.6) reduces to a self-attention block, $\text{MHAB}(\mathbf{X}, \mathbf{X})$, as commonly used in Set Transformers [49].

Decoder Block with Causal Masking (MHADB). The Transformer decoder augments a masked self-attention sublayer with cross-attention to an encoder context [48]. Given an *output* sequence $\mathbf{Y} \in \mathbb{R}^{t \times d}$ and an *input/encoder* sequence $\mathbf{X} \in \mathbb{R}^{n_x \times d}$,

$$\begin{aligned}\mathbf{H}' &= \text{LN}(\mathbf{Y} + \text{MHSA}_{\text{causal}}(\mathbf{Y})), \\ \mathbf{H} &= \text{LN}(\mathbf{H}' + \text{MHA}(\mathbf{H}', \mathbf{X}, \mathbf{X})), \\ \text{MHADB}(\mathbf{Y}, \mathbf{X}) &= \text{LN}(\mathbf{H} + \text{rFFN}(\mathbf{H})),\end{aligned}\tag{2.7}$$

where $\text{MHSA}_{\text{causal}}$ applies a time (causal) mask such that position u can only attend to positions $\leq u$. Classically in the context of next-token prediction, during training, teacher forcing is used by shifting $\mathbf{Y}$ right by one token, enabling a single forward pass. At test time, the decoder generates autoregressively, i.e., the output tokens are used as inputs at the next timestep.

Remarks. Eqs. (2.4)–(2.7) cover the attention primitives used throughout our models. In Chapter 4, we make extensive use of self-attention on sets (Eq. (2.5)) and cross-attention (Eq. (2.6)) to build AutoBot’s encoder, while decoder-style blocks (Eq. (2.7)) are employed for generating future multi-agent trajectories. In Chapter 5, we also make use of these transformer blocks for controllable multi-agent behaviour simulation. In Chapter 6, we show how these attention blocks can be used for vectorized latent diffusion models in generating novel road structures.

2.2 Latent Variable Models

Suppose one is interested in learning a generative model of data $\mathbf{x}$ which originates from the distribution $p(\mathbf{x})$. Latent variable models assume that the data is generated by some underlying random process with unobserved (latent) random variables $\mathbf{z}$. The latent variables are assumed to originate from some known prior distribution $p(\mathbf{z})$ (e.g., $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$). This results in the following joint distribution

$$p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}). \quad (2.8)$$

The latent variables $\mathbf{z}$ can be interpreted as providing the essence of what the corresponding datapoints $\mathbf{x}$ are.

We are generally interested in learning a deterministic mapping $f : \mathcal{Z} \times \Theta \rightarrow \mathcal{X}$ (with parameters $\theta \in \Theta$) such that the data likelihood is maximized. The generative process involves sampling $\mathbf{z} \sim p(\mathbf{z})$ and passing it through the deterministic mapping f_θ in order to generate the parameters of the distribution $p(\mathbf{x} \mid \mathbf{z})$ from which we can sample $\mathbf{x}$. Intuitively, we aim to maximize the probability of every datapoint $\mathbf{x}$ in the dataset under this generative process through the latents. The latent variables allow us to represent the complex distribution over the dataset $p(\mathbf{x})$ in terms of the simpler distributions $p(\mathbf{x} \mid \mathbf{z})$ and $p(\mathbf{z})$. The marginal likelihood of the data can be obtained by

$$p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) d\mathbf{z} \quad (\text{or } \sum_{\mathbf{z}} p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \text{ if } \mathbf{z} \text{ is discrete}), \quad (2.9)$$

which is intractable in most interesting problems due to the cardinality of $\mathbf{z}$ and the nonlinearity of f_θ .

If we follow the generative process above, we are not guaranteed to sample good values of $\mathbf{z} \sim p(\mathbf{z})$ that result in high-probability data. It is therefore necessary to refine the prior belief over the latent variables in order to sample relevant values of $\mathbf{z}$. This can be done by

computing the posterior distribution $p(\mathbf{z}|\mathbf{x})$ using Bayes' rule

$$p(\mathbf{z}|\mathbf{x}) = \frac{p(\mathbf{x}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{x})} \quad (2.10)$$

which allows us to infer the latent variables conditioned on the observations. However, as we mentioned above, $p(\mathbf{x})$ is intractable making Equation 2.10 also intractable. It is always possible to introduce a probability distribution $q(\mathbf{z})$ over the latent variables and transform the optimization problem into

$$q^*(\mathbf{z}) = \arg \min_{q(\mathbf{z}) \in \mathcal{Q}} \text{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] \quad (2.11)$$

However, this optimization problem is still intractable since it depends on $p(\mathbf{x})$. We can see this by using the definition of the Kullback–Leibler (KL) divergence and Bayes' rule,

$$\begin{aligned} \text{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] &= \mathbb{E}_q[\log q(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{z}|\mathbf{x})] \\ &= \mathbb{E}_q[\log q(\mathbf{z})] - \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z})] + \log p(\mathbf{x}) \end{aligned} \quad (2.12)$$

Moving the components around

$$\log p(\mathbf{x}) - \text{KL}[q(\mathbf{z})||p(\mathbf{z}|\mathbf{x})] = \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z})] \quad (2.13)$$

where the right-hand side is known as the evidence lower bound (ELBO). Since the minimum value of the $\text{KL}(\cdot||\cdot) \geq 0$, then maximizing the ELBO provides a lower bound on the maximum likelihood of the distribution $p(\mathbf{x})$.

Notice that the ELBO is equal to the log-likelihood of the data in the event that $q(\mathbf{z}) = p_\theta(\mathbf{z}|\mathbf{x})$ (since the KL would equal 0). The Expectation-Maximization (EM) [31] algorithm provides a method to maximize the likelihood of the data with latent variable models using this fact. Latent variable models output the parameters of the data distribution conditioned on the latent variables, $p_\theta(\mathbf{x}|\mathbf{z})$. Suppose the current parameters of the model are denoted by θ_{old} . In the E-Step, we substitute $q(\mathbf{z}) = p_{\theta_{old}}(\mathbf{z}|\mathbf{x})$ noting that this distribution is easy to compute analytically for certain models (e.g., Gaussian mixture models) using $p_{\theta_{old}}(\mathbf{x}|\mathbf{z})$ and $p(\mathbf{z})$. This gives us the following expression

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_q[\log p_\theta(\mathbf{x}, \mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z})] \\ &= \sum_{\mathbf{z}} p_{\theta_{old}}(\mathbf{z}|\mathbf{x}) \log p_\theta(\mathbf{x}, \mathbf{z}) - \sum_{\mathbf{z}} p_{\theta_{old}}(\mathbf{z}|\mathbf{x}) \log p_{\theta_{old}}(\mathbf{z}|\mathbf{x}) \\ &= Q(\theta, \theta_{old}) + H_{\theta_{old}}(\mathbf{z}|\mathbf{x}) \end{aligned} \quad (2.14)$$

where $H(\cdot|\cdot)$ is the entropy of the distribution and is constant with respect to θ . Notice that $Q(\theta, \theta_{old})$ is the *expectation* of the posterior with respect to the old parameters θ_{old} . The M-step then consists of *maximizing* $Q(\theta, \theta_{old})$ with respect to the parameters θ . Intuitively, the EM algorithm first makes the KL divergence term equal to zero (E-step) using the old parameters θ_{old} resulting in $\log p(\mathbf{x}) = \text{ELBO}$. Then, it optimizes the ELBO using the new parameters but fixing $q(\mathbf{z}) = p_{\theta_{old}}(\mathbf{z}|\mathbf{x})$. These steps are repeated until convergence is reached. The model we propose in Chapter 4 uses discrete latent variables and the EM Algorithm to model the multiple possible futures of agents.

2.2.1 Variational Inference and VAEs

For many problems, it is not possible to compute $p_{\theta_{old}}(\mathbf{z}|\mathbf{x})$ analytically. Variational inference offers an alternative. We first note that it is possible to re-write the ELBO to reveal the variational inference objective [51]:

$$\begin{aligned} \text{ELBO} &= \mathbb{E}_q[\log p(\mathbf{z})] + \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z})] \\ &= \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] - \text{KL}[q(\mathbf{z})||p(\mathbf{z})] \end{aligned} \tag{2.15}$$

Here, we see that in order to maximize the data likelihood, we should generate likely data using $\mathbf{z} \sim q(\mathbf{z})$ and minimize the KL divergence between $q(\mathbf{z})$ and $p(\mathbf{z})$. Using amortized inference [52], Variational Autoencoders (VAE) [53] use a parameterized inference model (also known as the encoder), $q_\phi(\mathbf{z}|\mathbf{x})$. Hence, in VAEs, the training procedure first encodes $\mathbf{x}$ into $\mathbf{z}$ using the encoder, and employs a decoder $p_\theta(\mathbf{x}|\mathbf{z})$ to output the same data point. Then, the objective in Equation 2.15 reduces to a reconstruction loss of the data with an additive KL regularizer on the latent distribution.

2.2.2 Diffusion Models

Diffusion probabilistic models and related approaches [54, 55, 56] have come to the forefront of generative modelling. At their core, these models are inspired by non-equilibrium thermodynamics in which they define a parameterized Markov chain trained using variational inference to produce samples from a data distribution. Our goal in this section is to present the foundations of deep denoising diffusion models (DDPMs) proposed by Ho et al. [56], which builds on [54, 55].

Diffusion models [54] are an instance of latent variable models where the data distribution is defined by $q(x_0) = \int q(x_{0:T})dx_{1:T}$ and the latent variables are all the intermediate x_t values in the Markov chain. Notably, these latent random variables are of the same dimensionality

as the data itself x_0 . The central idea behind diffusion models is that the generative process is defined by a forward process and reverse process. The approximate posterior in diffusion models is a fixed Markov chain, defined as **the forward process**, that adds small amounts of Gaussian noise to the data in T steps following a specific variance schedule $\beta_1, \dots, \beta_T$, which is fixed a-priori in DDPM [56]. Mathematically, the forward process is expressed as

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) := \prod_{t=1}^T q(\mathbf{x}_t \mid \mathbf{x}_{t-1}), \quad (2.16)$$

where $q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1 - \beta_t} \mathbf{x}_{t-1}, \beta_t \mathbf{I})$.

The β_t values are such that as $T \rightarrow \infty$, the latent x_T becomes distributed according to an isotropic Gaussian. Thanks to the defined forward process, it is noteworthy that one could sample x_t at an arbitrary timestep t in closed form,

$$q(\mathbf{x}_t \mid \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t} \mathbf{x}_0, (1 - \bar{\alpha}_t) \mathbf{I}), \quad (2.17)$$

where we define: $\alpha_t := 1 - \beta_t$, $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$

This is easily shown by exploiting the property of combining multiple Gaussian distributions

$$\begin{aligned} \mathbf{x}_t &= \sqrt{\alpha_t} \mathbf{x}_{t-1} + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon}_{t-1} \\ &= \sqrt{\alpha_t \alpha_{t-1}} \mathbf{x}_{t-2} + \sqrt{1 - \alpha_t \alpha_{t-1}} \bar{\boldsymbol{\epsilon}}_{t-2} \\ &\vdots \\ &= \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \end{aligned} \quad \text{where } \boldsymbol{\epsilon}_{t-1}, \boldsymbol{\epsilon}_{t-2}, \dots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \quad (2.18)$$

noting that $\bar{\boldsymbol{\epsilon}}_t$ is the result of combining two Gaussians with different variances. **The reverse process** is also defined by a Markov chain with *learned* Gaussian transitions with parameters θ ,

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^T p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \quad (2.19)$$

where $p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$, $p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t))$

It is important to note that the true one-step reverse posterior, $q(\mathbf{x}_{t-1} \mid \mathbf{x}_t)$ is not tractable in general as it requires access to the entire data distribution $q(x_0)$. However, if we condition

on x_0 , the reverse process is tractable and Gaussian,

$$q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I}), \quad \tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t, \quad (2.20)$$

with mean

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{\sqrt{\alpha_t} (1 - \bar{\alpha}_{t-1})}{1 - \bar{\alpha}_t} \mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}} \beta_t}{1 - \bar{\alpha}_t} \mathbf{x}_0. \quad (2.21)$$

Substituting $x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1 - \bar{\alpha}_t} \epsilon_t)$ (from isolating x_0 in Equation 2.18), we can re-write the mean in terms of x_t only,

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_t \right) \quad (2.22)$$

Given the forward (approximate posterior) $q(\mathbf{x}_{1:T} | \mathbf{x}_0)$ and the learned reverse model $p_\theta(\mathbf{x}_{0:T})$ in (2.19), variational inference yields

$$\begin{aligned} \log p_\theta(\mathbf{x}_0) &= \log \int \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T} | \mathbf{x}_0)} q(\mathbf{x}_{1:T} | \mathbf{x}_0) d\mathbf{x}_{1:T} \\ &\geq \mathbb{E}_q[\log p_\theta(\mathbf{x}_{0:T}) - \log q(\mathbf{x}_{1:T} | \mathbf{x}_0)] =: -\mathcal{L}_{\text{ELBO}}. \end{aligned} \quad (2.23)$$

where the last inequality is thanks to Jensen's inequality. Expanding the two factorizations and regrouping terms gives the standard KL decomposition

$$\mathcal{L}_{\text{ELBO}} = \mathbb{E}_q \left[\underbrace{\text{KL}(q(\mathbf{x}_T | \mathbf{x}_0) \| p(\mathbf{x}_T))}_{\mathcal{L}_T} + \sum_{t=2}^T \underbrace{\text{KL}(q(\mathbf{x}_{t-1} | \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} | \mathbf{x}_t))}_{\mathcal{L}_{t-1}} - \underbrace{\log p_\theta(\mathbf{x}_0 | \mathbf{x}_1)}_{\mathcal{L}_0} \right]. \quad (2.24)$$

Ho et al. [56] propose learning to predict the noise $\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$ and infer

$$\hat{\mathbf{x}}_0^\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{\bar{\alpha}_t}} \left(\mathbf{x}_t - \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right), \quad \boldsymbol{\mu}_\theta(\mathbf{x}_t, t) = \tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t, \hat{\mathbf{x}}_0^\theta(\mathbf{x}_t, t)), \quad (2.25)$$

and choose a fixed variance $\boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t) = \sigma_t^2 \mathbf{I}$, which is either $\sigma_t^2 = \tilde{\beta}_t$ or β_t . With these choices, each KL term in (2.24) reduces (up to a constant) to a weighted MSE:

$$\mathcal{L}_{t-1} \propto \mathbb{E}_q \left[\lambda_t \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2 \right], \quad \mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), \quad (2.26)$$

with weight

$$\lambda_t = \frac{\beta_t^2}{2 \sigma_t^2 \alpha_t (1 - \bar{\alpha}_t)}. \quad (2.27)$$

Finally, Ho et al. [56] propose ignoring constants $\mathcal{L}_T$ and replacing $\mathcal{L}_0$ with a simple Gaussian term, which reduces DDPM’s training to minimizing the “simple” objective below

$$\mathcal{L}_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q, t \sim \mathcal{U}\{1, \dots, T\}, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta \left(\sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}, t \right) \right\|^2 \right], \quad (2.28)$$

which is equivalent to minimizing the ELBO in (2.24) under the variance choices above (up to constants and per-step weights). We provide the training and sampling algorithms of DDPM in Algorithms 1 and 2.

Algorithm 1 DDPM Training

```

1: Inputs: variance schedule  $\{\beta_t\}_{t=1}^T$ , with  $\alpha_t := 1 - \beta_t$  and  $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$ ; network  $\boldsymbol{\epsilon}_\theta(\cdot, t)$ 
2: while not converged do
3:   Sample data  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ 
4:   Sample timestep  $t \sim \text{Uniform}\{1, \dots, T\}$ 
5:   Sample noise  $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
6:   # Form noisy input  $\mathbf{x}_t = \sqrt{\bar{\alpha}_t} \mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}$ 
7:   # Gradient step on the simple loss
8:    $\theta \leftarrow \theta - \eta \nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right\|^2$ 
9: end while
```

Algorithm 2 DDPM Sampling

```

1: Inputs:  $\{\beta_t\}_{t=1}^T$ ,  $\alpha_t := 1 - \beta_t$ ,  $\bar{\alpha}_t := \prod_{s=1}^t \alpha_s$ , choose  $\sigma_t^2 \in \{\tilde{\beta}_t, \beta_t\}$  where  $\tilde{\beta}_t = \frac{1 - \bar{\alpha}_{t-1}}{1 - \bar{\alpha}_t} \beta_t$ 
2: Initialize  $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ 
3: for  $t = T, T - 1, \dots, 1$  do
4:   Sample  $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$  if  $t > 1$ , else set  $\mathbf{z} = \mathbf{0}$ 
5:    $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left( \mathbf{x}_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$ 
6: end for
7: return  $\mathbf{x}_0$ 
```

One important problem with DDPMs is their heavy computational demands since the latents and all computations are performed in the high-dimensional space of RGB images. Rombach et al. [57] propose a two-stage training paradigm where one first trains a (variational) autoencoder on the original image to obtain a compressed representation, z_0 . Then, given the compressed representation, they train a *latent* diffusion model making the whole process more efficient. At sampling time, we first sample from a Gaussian distribution the initial random latents, $z_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and perform T steps of reverse diffusion until we obtain z_0 . The autoencoder’s decoder is then used to obtain the sample from our data distribution. We make use of latent diffusion models in the context of initial map and agent generation in our Scenario Dreamer model, presented in Chapter 6.

2.3 Reinforcement Learning

Reinforcement learning (RL) [58] is a framework for automatically learning control policies that satisfy an objective in sequential decision problems. The control problem in RL consists of solving a Markov Decision Process (MDP) (S, A, P, R, γ) where:

- S is the state-space;
- A is the action-space available to the agent;
- P is the state transition probability function of the system, $P(s_{t+1}|s_t, a_t)$;
- R is the reward function that generates a scalar reward as a function of the state-action pair, $r_t = R(s_t, a_t)$;
- $\gamma \in [0, 1]$ is a discount factor.

The RL problem simplifies the control problem as the interface between the agent and its environment, as portrayed in Figure 2.1. At every timestep t , the agent samples an action a_t according to its stochastic policy $\pi(a_t|s_t)$. The environment reacts by producing the agent's next state, s_{t+1} , and the next reward, $r_{t+1} = R(s_t, a_t)$.

The return, G_t , is defined as the sum of discounted rewards along a trajectory,

$$G_t = \sum_{k=0}^T \gamma^k r_{t+1+k} \quad (2.29)$$

The goal of reinforcement learning algorithms is to find the optimal policy, $\pi^*(a_t|s_t)$ that maximizes G_t . Notice that for $\gamma = 0$, the agent becomes myopic, only caring about maximizing the immediate reward, while $\gamma = 1$, the agent becomes farsighted. See Sutton and Barto [58] for a more in-depth discussion on the discount factor's usefulness in infinite horizon problems.

2.3.1 Value Functions

All reinforcement learning algorithms need to estimate the value of a state in order to improve its policy. The value of a state, $V^\pi(s)$, is defined as the expected sum of rewards the agent will receive if it follows policy π in state s .

$$V^\pi(s) = \mathbb{E}\left[\sum_{k=0}^T \gamma^k r_{t+1+k} \mid \pi\right] \quad (2.30)$$

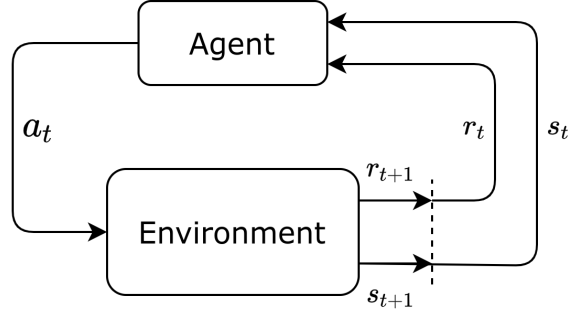


Figure 2.1 The Agent-Environment Interface in Reinforcement Learning.

A fundamental property of value functions is that they have a recursive relationship,

$$V^\pi(s) = \sum_a \pi(a|s) \sum_{s'} P(s'|s, a) [R(s, a) + \gamma V^\pi(s')] \quad (2.31)$$

This is also known as the Bellman equation for $V^\pi(s)$. Similarly, the value of a state-action pair, $Q^\pi(s, a)$, expresses the expected sum of rewards for being in state s , taking action a and following policy π thereafter. It can be expressed as

$$Q^\pi(s, a) = \sum_{s'} P(s'|s, a) [R(s, a) + \gamma V^\pi(s')] \quad (2.32)$$

We are interested in learning the value function from raw experience collected under a given policy. One way to achieve this is to use a Monte-Carlo approach where the value of every state is updated based on the full return obtained, which can only be computed at the end of the episode. As stated by Sutton and Barto [58], a more novel way (and central to all RL methods) is to use *temporal difference* (TD) learning, where one bootstraps the estimate of the state value using the old state value function estimates, i.e.,

$$V(s_t) \leftarrow V(s_t) + \alpha \delta_t \quad (2.33)$$

where $\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$ is the TD error, α is the step size, and $V(s)$ is our current state value function approximation of the true value function, $V^\pi(s)$, for policy π . A very similar equation can be written out for the TD error of the state-action value function.

The advantage function is an important quantity that indicates how much better it is to perform action a in state s rather than taking an action according to the policy π . It is

defined as the difference between the state-action value and the state value,

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s) \quad (2.34)$$

This quantity will be useful for our introduction to Proximal Policy Optimization below.

2.3.2 Policy Gradient Methods

In the previous section, we presented value functions which do not directly provide a policy but can be used to infer one by choosing the action that maximizes the value in a given state (e.g., ϵ -greedy policy). However, it is often desirable to learn a parameterized policy, $\pi_\theta(a|s)$, directly from experience without the need to consult a value function. A major advantage to directly learning a parameterized policy is that it can naturally be stochastic (a Softmax policy). This can lead to a smoother learning procedure as action probabilities would smoothly change during training, rather than an abrupt change due to state-action value shifts. In addition, learning a policy directly is more amenable to continuous action spaces.

An important theoretical result is the policy gradient theorem (PGT), which expresses the gradient of the objective function (to maximize the expected return) with respect to the policy parameters. We refer the reader to Chapter 9 of Sutton and Barto’s Introduction to Reinforcement Learning book [58] for a proof of the policy gradient.

The direct implementation of the PGT leads to the REINFORCE gradient estimate for policy parameters:

$$\nabla J(\theta) = \nabla_\theta G_t \log \pi_\theta(a_t|s_t) \quad (2.35)$$

Intuitively, this expression shows that we should change the parameters of the policy in the direction that maximizes the expected return G_t . However, this estimate of the gradient is known to have high variance as it can only be estimated once G_t can be computed. For this reason, it is known as a Monte Carlo approach and can be slow to learn good policy parameters. Using a baseline function that varies with the state (but not the action) to compare the return to has been shown to reduce the variance. An obvious choice is to use the state value function, which would provide an indication of how well the agent is doing compared to the (estimated) average value of the state. This would change Equation 2.35 to

$$\nabla J(\theta) = \nabla_\theta (G_t - V(s_t)) \log \pi_\theta(a_t|s_t) \quad (2.36)$$

Actor-critic methods combine the bootstrapping of temporal difference learning with policy gradients by replacing $G_t - V(s_t)$ in Equation 2.36 with $r_{t+1} + \gamma V(s_{t+1}) - V(s_t)$, i.e., the TD error of a learned state value function. As is the case with all TD methods, this increases bias while reducing the variance of the value estimates. Actor-Critic methods have dominated the field of RL, achieving state-of-the-art on many tasks [59, 60, 61, 62]. Next, we present a few actor-critic approaches which we employ in future chapters.

Proximal Policy Optimization

Proximal policy optimization (PPO; [62]) is an on-policy method building on Trust Region Policy Optimization (TRPO; [63]). The main idea behind TRPO is to improve the training stability of policy gradients by ensuring that the policy does not change too fast between training iterations. This is carried out by applying a KL constraint on the policy before and after the update. The surrogate objective function is

$$L(\theta) = \max_{\theta} \mathbb{E} \left[\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \hat{A}_t - \beta \text{KL}[\pi_{\theta}(\cdot|s_t), \pi_{\theta_{old}}(\cdot|s_t)] \right] \quad (2.37)$$

where β is a hyperparameter, $\hat{A}_t$ is the advantage function, θ_{old} and θ are the parameters before and after the update, respectively. The main idea is to ensure that the ratio $r_t(\theta) = \frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \approx 1$. However, the experiments in TRPO revealed that it is difficult to choose the value of β that performs well across tasks. PPO proposes to change this objective slightly to make it more stable and efficient to solve:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}[\min\{r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t\}] \quad (2.38)$$

In addition to the policy learning, in order to compute the advantage, it is necessary to learn the state value function. Using Generalized Advantage Estimation (GAE; [64]), for a fixed length T trajectory segment, the advantage is estimated by

$$\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + \dots + (\gamma\lambda)^{T-t+1}\delta_{T-1} \quad (2.39)$$

where δ_i is defined in Equation 2.33, γ is the discount factor and λ determines how much one wishes to bootstrap on later values. With this formulation of the Advantage function, one only needs to learn a state value function. The final loss function in PPO is

$$L_t = \mathbb{E}[L_t^{\text{CLIP}}(\theta) - c_1(V_{\theta}(s_t) - V_t^{\text{targ}})^2 + c_2 S[\pi_{\theta}](s_t)] \quad (2.40)$$

where L_t^{CLIP} is defined in Equation 2.38, $S[\pi_\theta](s_t)$ measures the entropy of the policy, the regression target for learning the state value function is shown in the middle and c_1 and c_2 are hyperparameters controlling the relative importance of the components in the loss terms. The RL experiments we present in Chapter 6 all use PPO with the loss function shown in Equation 2.40 to learn agent policies.

Group Relative Policy Optimization

In their seminal work, Ouyang et al. [65] propose the usage of PPO to finetune large language models (LLMs) towards preference optimization. Specifically, after a round of supervised finetuning of a LLM on demonstration data, a reward model is trained to predict if a specific LLM-generated response (i.e., an action) would be preferred by a human labeler. This reward model is then used in an instance of the PPO algorithm presented in Section 2.3.2 where the environment is a bandit environment, i.e., the agent takes a single action, receives a reward, and the episode terminates. The objective function in this instance of PPO can be written as

$$\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{q \sim P(Q), o \sim \pi_{\theta_{\text{old}}}(o|q)} \left[\frac{1}{|o|} \sum_{t=1}^{|o|} \min \left(r_t(\theta) A_t, \text{clip}(r_t(\theta), 1 - \varepsilon, 1 + \varepsilon) A_t \right) \right], \quad (2.41)$$

$$\text{where } r_t(\theta) = \frac{\pi_\theta(o_t | q, o_{<t})}{\pi_{\theta_{\text{old}}}(o_t | q, o_{<t})}.$$

where A_t is computed using GAE (Equation 2.39) where the value function is replaced by the reward model.

One issue with this approach is the need for a value function, which in the language space, is also a large model, bringing with it substantial memory and computational costs. In addition, the reward model could be prone to over-optimization during training. To mitigate this, [65] proposed to add a per-token KL penalty from the reference model, where the reference model is the original one before training began. Specifically, the new reward function becomes

$$r_t = r_\phi(q, o_{\leq t}) - \beta \log \frac{\pi_\theta(o_t | q, o_{<t})}{\pi_{\text{ref}}(o_t | q, o_{<t})} \quad (2.42)$$

where r_ϕ is the reward model, π_{ref} is the reference model and β is the coefficient controlling how close we remain to it.

Operating with the assumption that one has access to a queryable reward function that provides scalar values to a query (e.g., BLEU/ROUGE, math checker), Shao et al. [66] proposed to get rid of the reward model altogether for advantage estimation. Instead, they

propose generating G candidate outputs $\{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(O \mid q)$ to a given question $q \sim P(Q)$ and perform the advantage estimation within the group using

$$\hat{A}_{i,t} = \frac{r_{i,t} - \text{mean}(\mathbf{r}_t)}{\text{std}(\mathbf{r}_t)} \quad (2.43)$$

Equipped with this advantage estimate, the PPO objective is modified to the group relative policy optimization (GRPO)

$$\begin{aligned} \mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q \sim P(Q), \{o_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(O \mid q)} & \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|o_i|} \sum_{t=1}^{|o_i|} \right. \\ & \left. \min \left(r_{i,t}(\theta) \hat{A}_{i,t}, \text{clip} \left(r_{i,t}(\theta), 1 - \varepsilon, 1 + \varepsilon \right) \hat{A}_{i,t} \right) - \beta \text{KL}(\pi_{\theta} \parallel \pi_{\text{ref}}) \right], \\ \text{where } r_{i,t}(\theta) = & \frac{\pi_{\theta}(o_{i,t} \mid q, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid q, o_{i,<t})}. \end{aligned} \quad (2.44)$$

where the KL objective has been modified to be between the trained policy and the reference policy directly, rather than being part of the reward. GRPO has shown great promise in mathematical reasoning [66] as well as in autonomous-driving-focused research [67, 68, 69]. We make use of GRPO in our experiments in Chapter 7.

Soft Actor-Critic

Soft Actor-Critic (SAC; [59]) is an off-policy, model-free actor-critic algorithm based on the maximum entropy reinforcement learning framework. It augments the classical RL objective by incorporating the entropy measure of the policy into the reward. The goal is to encourage stochasticity in the action distribution to support continual exploration while maximizing return. The maximum-entropy objective can be written as

$$J(\pi) = \sum_t \mathbb{E}_{(s_t, a_t) \sim \pi} [r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t))], \quad (2.45)$$

where $\alpha > 0$ is the temperature (i.e., entropy weight) which controls how important entropy is. In addition to exploration, the entropy maximization also incentivizes multimodality in the policy, i.e., if two actions are equally good, they should have equal likelihood under the

policy. The soft Q-value function can be computed using the modified Bellman backup,

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_{a \sim \pi} [Q^\pi(s, a) - \alpha \log \pi(a | s)], \\ Q^\pi(s, a) &= r(s, a) + \gamma \mathbb{E}_{s' \sim P} [V^\pi(s')]. \end{aligned} \quad (2.46)$$

SAC employs three sets of parameterized models: the state-value function $V_\psi(s_t)$, the action-value function $Q_\theta(s_t, a_t)$ and the policy $\pi_\phi(a_t | s_t)$. While in theory one can estimate the state value by sampling an action and querying the action-value function; however, making it a separate function stabilizes training [59]. The soft state value function is trained to minimize

$$J_V(\psi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\frac{1}{2} \left(V_\psi(s_t) - \mathbb{E}_{a_t \sim \pi_\phi} [Q_\theta(s_t, a_t) - \log \pi_\phi(a_t | s_t)] \right)^2 \right] \quad (2.47)$$

using the state and action samples from a replay buffer. The soft Q-function minimizes the soft Bellman residual

$$J_Q(\theta) = \mathbb{E}_{(s_t, a_t) \sim \mathcal{D}} \left[\frac{1}{2} \left(Q_\theta(s_t, a_t) - \left(r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p} [V_{\bar{\psi}}(s_{t+1})] \right) \right)^2 \right] \quad (2.48)$$

where $V_{\bar{\psi}}(s_{t+1})$ is a target value network, typically an exponentially moving average of the network weights ψ . Although it is not strictly necessary, SAC employs two sets of parameters (θ_1 and θ_2) of the Q-function to mitigate positive bias in the policy improvement step. Each function is trained independently to minimize Equation 2.48, and use the minimum of the two for training the state value function (Equation 2.47) and the policy in Equation 2.50. Finally, the policy is trained to minimize the policy improvement step, which follows

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} \left[\text{D}_{\text{KL}} \left(\pi_\phi(\cdot | s_t) \left\| \frac{\exp(Q_\theta(s_t, \cdot))}{Z_\theta(s_t)} \right) \right) \right] \quad (2.49)$$

Ensuring that $\pi \in \Pi$ is a set of policies that we can model tractably, such as a spherical Gaussian, we can rewrite the KL divergence as

$$J_\pi(\phi) = \mathbb{E}_{s_t \sim \mathcal{D}} [\log \pi_\phi(a_t | s_t) - Q_\theta(s_t, a_t)] \quad (2.50)$$

which ignores the partition function $Z_\theta(s_t)$ since it is intractable and does not contribute to the gradient with respect to the policy parameters. We provide the final algorithm for how one can train a policy using SAC in Algorithm 3. We employ a modified version of SAC in our experiments in Chapter 7.

Algorithm 3 Soft Actor-Critic

```

1: Initialize parameters  $\psi$ ,  $\bar{\psi}$ ,  $\theta$  and  $\phi$ .
2: Initialize replay buffer  $\mathcal{B}$ .
3: for updates  $u = 1, 2, \dots$  (until convergence) do
4:   # Data collection
5:   for each environment step do
6:     Sample from current policy:  $a \sim \pi_\phi(\cdot \mid s)$ 
7:     Interact with environment to get next state and reward:  $(s', r)$ 
8:     Append transition  $(s, a, r, s')$  to  $\mathcal{B}$ 
9:   end for
10:  # Policy gradient update
11:  for each gradient step do
12:     $\psi \leftarrow \psi - \lambda_V \hat{\nabla}_\psi J_V(\psi)$ 
13:     $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$  for  $i \in \{1, 2\}$ 
14:     $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ 
15:     $\bar{\psi} \leftarrow \tau\psi + (1 - \tau)\bar{\psi}$ 
16:  end for
17: end for

```

CHAPTER 3 LITERATURE REVIEW

In this chapter, we present the related work corresponding to the contributions of this thesis. We note that the literature review for multi-agent motion forecasting is not included here, as Chapter 4 is a chapter by article and already integrates its related work section. Instead, we focus on prior literature related to controllable data-driven behaviour simulation for our proposed approaches in Chapters 5 and 6, and constrained reinforcement learning for behaviour specification for the work in Chapter 7.

3.1 Data-Driven Simulation for Autonomous Driving

Data-driven simulation for autonomous driving can be naturally decomposed into two coupled subproblems: the generation of an initial scene and the modelling of agent behaviour over time. Formally, we consider the joint distribution

$$P(\mathbb{X}_{0:T}, \mathcal{M} \mid c) = P(\mathbb{X}_{1:T} \mid \mathcal{M}, \mathbb{X}_0, c) P(\mathcal{M}, \mathbb{X}_0 \mid c),$$

where $\mathcal{M}$ denotes the map, $\mathbb{X}_t$ the multi-agent state at time t , and c denotes conditioning information. The first term, $P(\mathcal{M}, \mathbb{X}_0 \mid c)$, corresponds to initial scene generation, including map structure and initial agent configurations, while the second term, $P(\mathbb{X}_{1:T} \mid \mathcal{M}, \mathbb{X}_0, c)$, models the evolution of agent behaviours through interactions in the environment. These components are typically embedded within a simulation environment that defines how scenes are represented, generated, and evolved over time. In this section, we review prior work along these axes, covering simulation environments, initial scene generation, and agent behaviour modelling, and discuss how they relate to the approaches proposed in this thesis.

3.1.1 Simulation Environments for Autonomous Driving

Driving simulators usually require a lot of manual effort in designing environments that are realistic and can support the rich interactions between multiple agents [8, 70, 71]. As a result, it has been difficult to scale to larger environments. In addition, due to the synthetic nature of the simulators, models tend to suffer from a significant sim-to-real gap. To address these issues, recent works have proposed data-driven driving simulators that are based on real driving data [72]. Prior data-driven approaches to simulation have fallen under two main approaches. The first focuses on reconstructing driving environments from sensor inputs using neural rendering techniques [73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83]. Other work instead

utilizes post-perception outputs to represent driving scenes in an abstract representation comprising agent bounding boxes and HD maps [38, 43, 84, 85, 86, 87, 88]. In this work, and in line with this thesis’ work, we adopt the abstract environment representation. This allows us to efficiently evaluate AV planners with sufficient complexity while having lower computational demands and memory footprint.

Generative Simulation Environments Open-source, data-driven simulators narrow the sim-to-real gap relative to hand-crafted engines. However, they remain constrained by two factors: the scarcity of public driving data and the short duration of most recorded logs. To mitigate these limits, recent work resorts to diffusion generative models to synthesize driving environments either directly from sensor inputs [89, 90, 91] or from post-perception abstract representations [92, 93]. Parallel efforts apply diffusion as a generator of interactive worlds in other domains, including video games [94, 95, 96] and robotics [97, 98, 99]. SLEDGE [93] adopts a tailored latent diffusion model to produce bounding boxes and lane graphs, and couples these with rule-based agent controllers (IDM, MOBIL), which limits behavioral realism [32, 33, 93]. DriveSceneGen [92] uses diffusion for initial scene synthesis and a motion-prediction model for agent behavior; however, the resulting scenarios are short and largely non-reactive (typically ≤ 20 seconds). Scenario Dreamer (from Chapter 6) addresses these shortcomings by pairing a novel vectorized latent diffusion model for initial scene generation with CtRL-Sim [26] from Chapter 5, yielding a fully data-driven, closed-loop generative driving simulator capable of arbitrarily long and interactive simulations.

3.1.2 Initial Scene Generation: Modelling $P(\mathcal{M}, \mathbb{X}_0 \mid c)$

Recent work focused specifically on the task of initial scene generation. Some target either the generation of initial agent bounding boxes configurations within a given map [100, 101], while others have focused solely on the generation of lane graphs [102]. More recent work has focused on modelling the joint initial distribution of both [92, 93]. SceneControl [101] utilizes a vectorized diffusion model for initial agent generation but does so in data space. In addition, it does not focus on the problem of vectorized HD map generation, and assumes this is provided. Both SLEDGE and DriveSceneGen employ diffusion models to generate the initial scene by processing it as a rasterized image, leading to unnecessary computation. Building on the rich recent literature on vectorized scene representations [103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115], in Chapter 6, we propose a novel vectorized latent diffusion model that generates the initial scene (vectorized map elements $\mathcal{M}$ and agent states $\mathbb{X}_0$), enhancing the speed and performance of the scene generation and extrapolations.

3.1.3 Agent Behaviour Simulation: Modelling $P(\mathbb{X}_{1:T} \mid \mathcal{M}, \mathbb{X}_0, c)$

We define agent behaviour simulation as the problem of modelling the behaviour of agents in simulation (e.g., pedestrians, cyclists, vehicles). The literature has explored two main approaches for behaviour simulation (1) rule-based modelling, and (2) data-driven approaches. As the name implies, rule-based approaches rely on heuristics, defined by humans, to produce plausible agent behaviours, such as adhering strictly to the center of the lane [32, 33]. However, such approaches can often produce rigid or unrealistic agent behaviors, and can fail to capture the different modalities of agent behaviours.

To address these limitations, recent research explored generative models that can reproduce the behaviors of real-world agents using large-scale trajectory datasets as supervision [27, 34, 35, 116, 117, 118, 119]. Many of these methods draw inspiration from the literature on joint motion prediction [18, 109, 111, 120, 121], where the task is to predict the future behaviour of multiple interacting agents. However, an important difference to consider is the fact that motion prediction is an open-loop problem, where one only needs to make predictions of the future; those predictions are not used to control the agents’ behaviours. This distinction is important because it means open-loop generated forecasts cannot affect the state of the environment [6].

To generate realistic simulated behaviours, other lines of work employed generative adversarial imitation learning (GAIL) [122], whose goal is to generate trajectories that are indistinguishable from expert demonstrations [123, 124, 125]. Other related approaches have additionally combined imitation approaches with reinforcement learning to enforce desirable properties, e.g., compliance with traffic rules [126, 127]. While these approaches all exhibit better realism compared to rule-based approaches, they still face a key limitation: their lack of controllability which is often necessary for useful simulated training and evaluation.

More recent work has focused on the problem of controllability with the goal of providing the user with a way to specify how agents should act. Conditional generative models have been explored in this regard [36, 128, 129, 130, 131, 132, 133] with the idea of incorporating different conditioning factors, allowing the user to produce diverse and conditioned behaviours at inference time. [128, 129] employ high-level latent variables that capture abstract driving styles. MixSim [130] conditions on explicit route information, defining a specific route for all agents. [36, 132, 133, 134, 135] all enforce safety or scenario-level requirements through differentiable constraints, and employ some form of iterative refinement. Another direction has introduced retrieval-augmented generation[136], drawing realistic examples from a database to generate controllable traffic scenarios.

While these methods represent a step towards more flexible traffic simulation, they come with their own set of drawbacks. Some lack interpretable control over the generated behaviours [129] or require costly test-time optimization procedures to steer the generated behaviours, such as latent variable optimization [128], Bayesian optimization [130, 137, 138], or the simulation of expensive diffusion processes [36, 115, 132, 133, 134, 135]. In contrast, CtRL-Sim (presented Chapter 5) introduces a more efficient procedure for controllability, without the need for iterative test-time optimization, while maintaining interpretable knobs on the returns. It achieves this by learning a conditional multi-agent behavior model that conditions directly on factorized returns. By applying an exponential tilting of the predicted return distribution [40] at test time, CtRL-Sim provides fine-grained, human-interpretable, and efficient control over agent behaviours, all while staying grounded in real-world data.

Return-Conditioned Policies and Offline Reinforcement Learning We position CtRL-Sim within the field of offline RL. Table 3.1 presents the different ways explored in the literature for learning policies in offline RL, and how these methods can sample return-maximizing actions at test time. Prior methods in offline RL differ in how the policy is modelled during training and how inference is performed; we refer the reader to Table 3.1 for a breakdown of the different approaches. As CtRL-Sim employs a return-conditioned policy for controllable simulation, we briefly present a class of related methods that learn return-conditioned policies (RCPs) [40, 41, 140, 141, 142, 143, 144] for offline RL. RCPs are concerned with learning the joint distribution of actions and returns, such that action sampling is conditioned on the return distribution. Instead of modelling the return distribution, the Decision Transformer [140] conditions the learned policy on the maximum observed return, $G_{\max}$ in the dataset. That is, at inference time, the actions are sampled from $a_t \sim p_\theta(a_t|s_t, G_{\max})$. Lee et al. [40] instead propose to employ the learned return dis-

Approach	Density Estimation	Action Sampling Density
Decision Transformers (DTs)	$\log p_\theta(a_t o_t, G_t)$	$p_\theta(a_t o_t, G_t)$
Reward Weighted Regression (RWR)	$\exp(\eta^{-1}G_t) \log p_\theta(a_t o_t)$	$p_\theta(a_t o_t)$
Reward Conditioned Policies (RCPs)	$\log p_\theta(a_t o_t, G_t)p_\theta(G_t o_t)$	$p_\theta(a_t o_t, G_t)p_\theta(G_t o_t) \exp(\kappa G_t - \eta(\kappa))$
Reweighted Behavior Cloning (RBC)	$\log p_\theta(G_t o_t, a_t)p_\theta(a_t o_t)$	$p_\theta(G_t o_t, a_t)p_\theta(a_t o_t) \exp(\kappa G_t - \eta(\kappa))$
Implicit RL via SL (IRvS)	$\log p_\theta(a_t, G_t o_t)$	$p_\theta(a_t, G_t o_t) \exp(\kappa G_t - \eta(\kappa))$
Model-Based RCPs (MB-RCP)	$\log p_\theta(o_{t+1} a_t, o_t, G_t)p_\theta(a_t o_t, G_t)p_\theta(G_t o_t)$	$p_\theta(a_t o_t, G_t)p_\theta(G_t o_t) \exp(\kappa G_t - \eta(\kappa))$
RCP with Future Rollout (CtRL-Sim)	$\log p_\theta(s_{t+1:T} a_t, s_t, G_t)p_\theta(a_t s_t, G_t)p_\theta(G_t s_t)$	$p_\theta(a_t s_t, G_t)p_\theta(G_t s_t) \exp(\kappa G_t - \eta(\kappa))$

Table 3.1 **Offline policy modelling approaches in prior work** [26]. We can see that methods differ in the decomposition of the joint distribution over actions and returns, with some approaches utilizing state prediction as a regularizer. We note that this table is adopted from prior work [41, 139].

tribution, combined with exponential tilting, in order to sample high-return actions while remaining close to the empirical distribution. CtRL-Sim adopts exponential tilting of the predicted return distribution for each agent to finely control agent behaviour. Additionally, CtRL-Sim explicitly models the future sequences of states.

3.2 Behaviour Specification Through Constrained RL

In Chapter 7, we propose a constrained RL framework for behaviour specification and extend it to driving with large vision-language models. In this section, we review prior work on constrained reinforcement learning, reward specification, and RL-based behaviour control in video games. We then discuss related work on constrained and multi-objective RL for large language models, with particular focus on how these methods express and enforce desired behaviours.

3.2.1 Constrained Reinforcement Learning

Constrained Markov Decision Processes (CMDPs) [145] provide a principled framework for reinforcement learning under explicit behavioural constraints. A large body of prior work has focused on developing algorithms that enforce constraints while optimizing for reward. Broadly, existing approaches can be divided into two families.

Lagrangian-based methods. Lagrangian approaches [146, 147, 148] incorporate the constraints into the main objective through a set of multipliers, forming a single Lagrangian function. Learning then reduces to finding a saddle point of this function, corresponding to a feasible solution to the constrained maximization problem. These methods are conceptually simple and widely used, but typically rely on dual ascent schemes that may introduce instability, especially when dealing with multiple constraints or high-variance gradient estimates.

Projection-based methods. An alternative line of work [149, 150, 151, 152] uses projection operators to map the updated policy back into the feasible region after performing a reward-maximization step. Such methods provide more direct feasibility guarantees under certain assumptions, but often require carefully designed trust regions or Lyapunov functions to remain stable.

Most prior work on constrained RL focuses on the *single-constraint* setting [148, 152, 153, 154] and typically seeks to minimize cumulative cost regret over the course of training [155]. In contrast, our work focuses on leveraging CMDPs as a *behaviour specification mechanism*,

allowing for multiple simultaneously enforced constraints that are both interpretable and easily configurable.

3.2.2 Reward Specification

A central motivation for our work is the difficulty of specifying complex behaviours through a single reward function. This challenge has inspired several alternative approaches in the literature.

Imitation and human-in-the-loop methods. Imitation Learning [156] bypasses reward engineering entirely by using expert demonstrations to define the task. Other methods introduce human feedback during training, either to guide the agent toward desired behaviours [157] or to prevent catastrophic errors during exploration [158]. Our approach shares a similar motivation—capturing designer intent more directly—but does so by specifying constraints via indicator functions and thresholds, avoiding the need for demonstrations or continuous human intervention.

Natural language as a specification interface. Another promising direction is the use of natural language to specify tasks [159, 160]. While this can provide a more intuitive interface for human designers, language is inherently ambiguous and can be exploited by agents (reward hacking). Moreover, these methods typically require training an additional language-to-reward model, adding further complexity.

Inverse reward design. Another interesting approach is Inverse Reward Design [161, 162, 163], which treats the provided reward function as a single observation of the designer’s latent intent and learns a probabilistic model over possible true reward functions. This is particularly useful in settings with changing environments, but our work focuses on fixed-distribution environments where explicit behaviour specification is more appropriate.

3.2.3 Reinforcement Learning in Video Games

Video games have long served as challenging benchmarks for deep RL research [164, 165, 166]. However, the use of RL in actual game production remains relatively rare. This gap stems from multiple practical considerations, including the difficulty of shaping desired behaviours, challenges in interpretability, and strict compute constraints during runtime [167, 168].

More recently, there has been growing interest in leveraging RL to power non-player characters (NPCs) in commercial games. Applications include navigation [168, 169], automated testing [170, 171], play-style modeling [172], and content generation [173]. Our work contributes to this emerging area by proposing a constrained RL framework that enables *direct behaviour specification*, making RL-controlled agents more controllable and predictable—key properties for production use.

3.2.4 Constrained RL for LLMs

Multiple objectives with (vision-)language models. In the context of large (multi-modal) language models, there is extensive work on safety and multi-objective optimization via reinforcement learning [174, 175, 176, 177, 178]. For example, Constitutional AI [178] can be viewed as optimizing a multi-objective alignment target, most notably balancing helpfulness and harmlessness. In practice these signals are reduced to a single scalar reward and optimized with standard RLHF pipelines [65], e.g., PPO-style updates [62]. Moving beyond fixed scalarization, Moskovitz et al. [174] frame composite reward models in RLHF as a constrained optimization problem, enforcing per-component thresholds via a CMDP formulation and using dynamic weight adjustment (i.e., Lagrange multipliers) to mitigate over-optimization. Similarly, Dai et al. [176] cast alignment as maximizing reward subject to explicit cost constraints, formalizing the trade-off between helpfulness and safety-related costs. Wang et al. [179] study multi-objective reward modeling in RLHF, emphasizing interpretability as a tool for designers to verify and debug adherence to human preferences.

Multi-Reward Optimization with GRPO. While the approaches above address multi-objective alignment for RL-finetuned (vision-)language models, they are largely developed outside the GRPO setting. GRPO [66] has recently become a popular alternative to PPO for RLHF-style fine-tuning, motivating a wave of follow-up work that analyzes and modifies its normalization and optimization biases. Dr.GRPO [180] identified that due to the standard deviation term, GRPO would artificially increase response lengths during training in the domain of reasoning. Their proposed approach alleviates these issues by removing the per-group standard deviation term, but does not explicitly focus on the problem of multi-objective optimization. DisCO [181] analyzes GRPO under binary rewards and shows that group-relative normalization induces a difficulty-dependent reweighting (down-weighting examples that are too easy or too hard). The authors propose a discriminative optimization framework whereby they naturally increase the scores of positive answers while reducing the scores of negative answers through a log-sum-exp function.

In the multi-objective GRPO domain, Li et al. [182] train a multi-head reward model predicting $K=4$ attributes (politeness, meaningfulness, actionability, safety), aggregate these scores into a single scalar reward via a fixed weighted sum (with $w_k=1$), and then optimize this combined objective with GRPO—mirroring the common practice of *scalarizing rewards prior to normalization*. Following reward model training and combination, the final policy is trained to maximize the combined scalar reward using GRPO, akin to Scalarizing rewards as shown in Section 7.4.3. We note that recent prior and concurrent work [183, 184, 185] has already observed that, under multiple objectives, GRPO’s normalization makes it difficult to learn from components with low standard deviation. To address multi-objective pathologies, Ichihara et al. [183] propose Multi-Objective Group Relative Policy Optimization (MO-GRPO), focusing on mitigating reward-hacking behavior of GRPO in the presence of multiple objectives: they show that GRPO’s advantage is more strongly correlated with reward components that exhibit higher variance, and advocate *scalarized advantages* so that components with different variances are scaled to have equal influence. However, MO-GRPO does not account for the consequences of a weighted-combination pipeline, which Theorem 7.4.1 shows can greatly attenuate their intended effects; this attenuation renders the Lagrangian multipliers nearly useless. Concurrent to our work, GDPO [185] also supports advantage scalarization and emphasizes that naïve GRPO normalization can collapse multi-reward signals, proposing reward-decoupled (and conditional) objectives to better preserve per-reward learning signals. Plan-R1 [184] proposes a planning objective with GRPO in which the reward is multiplicative, effectively imposing a hard constraint on behavior (see Section 7.5.2 for details). They further highlight that group-wise normalization can erase cross-group scale differences, but do not provide a mathematical analysis of the issue. Similarly to Dr.GRPO, they propose Variance-Decoupled GRPO, which changes advantage estimation to divide by a constant value rather than the standard deviation. In Chapter 7, our goal is to ensure that increasing the Lagrangian multiplier in our proposed Constrained GRPO algorithm faithfully results in stronger enforcement of the behavior specification; as we show experimentally, *scalarizing advantages* provides this guarantee compared to the traditional approach of *scalarizing rewards*.

In contrast to prior GRPO variants that primarily target stability or reward-hacking, we study *advantage scalarization through the lens of constrained policy optimization*. We show that *scalarizing rewards* before GRPO’s group normalization can implicitly reweight objectives as a function of within-group variances and cross-term correlations, distorting the prescribed priorities encoded by the multipliers. By contrast, *scalarizing advantages* preserves the intended role of the multipliers and yields a coherent learning signal for constrained optimization.

CHAPTER 4 ARTICLE 1: LATENT VARIABLE SEQUENTIAL SET TRANSFORMERS FOR JOINT MULTI-AGENT MOTION PREDICTION

Roger Girgis^{1,2}, Florian Golemo^{2,3}, Felipe Codevilla^{2,4}, Martin Weiss^{1,2}, Jim Aldon D’Souza⁵, Samira E. Kahou^{2,6,7,8}, Felix Heide^{5,9}, Christopher Pal^{1,2,3,8}

¹Polytechnique Montréal; ²Mila, Quebec AI Institute; ³ElementAI / ServiceNow; ⁴Independent Robotics; ⁵Algolux; ⁶McGill University; ⁷École de technologie supérieure; ⁸Canada CIFAR AI Chair; ⁹Princeton University

Presentation. This work was published in the proceedings of the International Conference on Learning Representations (ICLR) in January 28th, 2022.

4.1 Abstract

Robust multi-agent trajectory prediction is essential for the safe control of robotic systems. A major challenge is to efficiently learn a representation that approximates the true joint distribution of contextual, social, and temporal information to enable planning. We propose Latent Variable Sequential Set Transformers which are encoder-decoder architectures that generate scene-consistent multi-agent trajectories. We refer to these architectures as “AutoBots”. The encoder is a stack of interleaved temporal and social multi-head self-attention (MHSA) modules which alternately perform equivariant processing across the temporal and social dimensions. The decoder employs learnable seed parameters in combination with temporal and social MHSA modules allowing it to perform inference over the entire future scene in a single forward pass efficiently. AutoBots can produce either the trajectory of one ego-agent or a distribution over the future trajectories for all agents in the scene. For the single-agent prediction case, our model achieves top results on the global nuScenes vehicle motion prediction leaderboard, and produces strong results on the Argoverse vehicle prediction challenge. In the multi-agent setting, we evaluate on the synthetic partition of TrajNet++ dataset to showcase the model’s socially-consistent predictions. We also demonstrate our model on general sequences of sets and provide illustrative experiments modelling the sequential structure of the multiple strokes that make up symbols in the Omniglot data. A distinguishing feature of AutoBots is that all models are trainable on a single desktop GPU (1080 Ti) in under 48h.

4.2 Introduction

Many problems require processing complicated hierarchical compositions of sequences and sets. For example, multiple choice questions are commonly presented as a query (e.g. natural language) and an unordered set of options [186, 187]. Machine learning models that perform well in this setting should be insensitive to the order in which the options are presented. Similarly, for motion prediction tasks [188, 189] where the inputs are agent trajectories evolving over time and outputs are future agent trajectories, models should be insensitive to agent ordering.

In this work, we focus on the generative modelling of sequences and sets and demonstrate this method on a variety of different motion forecasting tasks. Suppose we have a set of $M \in \mathbb{N}$ sequences $\mathbb{X} = \{(x_1, \dots, x_K)_1, \dots, (x_1, \dots, x_K)_M\}$ each with $K \in \mathbb{N}$ elements. Allowing also that $\mathbb{X}$ evolves across some time horizon T , we denote this sequence of sets as $\mathbf{X} = (\mathbb{X}_1, \dots, \mathbb{X}_T)$.

This problem setting often requires the model to capture high-order interactions and diverse futures. Generally, solutions are built with auto-regressive sequence models which include t steps of context. In our approach, to better model the multi-modal distribution over possible futures, we also allow to include a discrete latent variable Z , to create sequential models of sets of the form

$$P(\mathbb{X}_{t+1}|\mathbb{X}_t, \dots, \mathbb{X}_1) = \sum_Z P(\mathbb{X}_{t+1}, Z|\mathbb{X}_t, \dots, \mathbb{X}_1).$$

In this work, we propose this type of parameterized conditional mixture model over sequences of sets, and call this model a Latent Variable Sequential Set Transformer (affectionately referred to as “AutoBot”). We evaluate AutoBot on *nuScenes* [190] and *Argoverse* [191], two autonomous driving trajectory prediction benchmarks, on the synthetic partition of the *TrajNet++* [192] dataset, a pedestrian trajectory forecasting benchmark, and *Omniglot* [193], a dataset of hand-drawn characters that we use for predicting strokes. All of these provide empirical validation of our model’s ability to perform robust multi-modal prediction. Specifically, we make the following contributions:

- We propose a novel approach for modelling sequences of set-structured continuous variables, and extend this to a latent-variable formulation to capture multi-modal distributions.
- We provide a theoretical analysis highlighting our model’s permutation sensitivity with respect to different components of the input data structures.
- We validate our method with strong empirical results on diverse tasks, including the

large-scale *nuScenes* and *Argoverse* datasets for autonomous driving [190, 191], the multi-lingual *Omniglot* dataset of handwritten characters [193], and the synthetic partition of the *TrajNet++* pedestrian trajectory prediction task [189].

4.3 Background

We now review several components of the Transformer [48] and Set Transformer [49] architectures, mostly following the notation found in their manuscripts. For additional background, see their works and Section 2.1 of the background chapter.

Multi-Head Self Attention (MHSA) can be thought of as an information retrieval system, where a query is executed against a key-value database, returning values where the key matches the query best. While [48] defines MHSA on three tensors, for convenience, we input a single set-valued argument. Internally, MHSA then performs intra-set attention by casting the input set $\mathbb{X}$ to query, key and value matrices and adding a residual connection, $\text{MHSA}(\mathbb{X}) = \mathbf{X} + \text{MHSA}(\mathbf{X}, \mathbf{X}, \mathbf{X})$.

Multi-Head Attention Blocks (MAB) resemble the encoder proposed by Vaswani et al. [48], but lack the position encoding and dropout [49]. Specifically, they consist of the MHSA operation described in Eq. 2.5 followed by a row-wise feed-forward neural network (rFFN), with residual connections and layer normalization (LN) [50] after each block. Given an input set $\mathbb{X}$ containing n_x elements of dimension d , and some conditioning variables $\mathbb{C}$ containing n_c elements of dimension d , the MAB can be described by the forward computation,

$$\text{MAB}(\mathbf{X}) = \text{LN}(\mathbf{H} + \text{rFFN}(\mathbf{H})), \quad \text{where} \quad \mathbf{H} = \text{LN}(\mathbf{X} + \text{MHSA}(\mathbf{X}, \mathbf{X}, \mathbf{X})). \quad (4.1)$$

Multi-head Attention Block Decoders (MABD) were also introduced in [48], and were used to produce decoded sequences. Given input matrices $\mathbf{X}$ and $\mathbf{Y}$ representing sequences, the decoder block performs the following computations:

$$\begin{aligned} \text{MABD}(\mathbf{Y}, \mathbf{X}) &= \text{LN}(\mathbf{H} + \text{rFFN}(\mathbf{H})) \\ \text{where} \quad \mathbf{H} &= \text{LN}(\mathbf{H}' + \text{MHSA}(\mathbf{H}', \mathbf{X}, \mathbf{X})) \\ \text{and} \quad \mathbf{H}' &= \text{LN}(\text{MHSA}(\mathbf{Y})) \end{aligned} \quad (4.2)$$

Our model, described throughout the following section, makes extensive use of these functions.

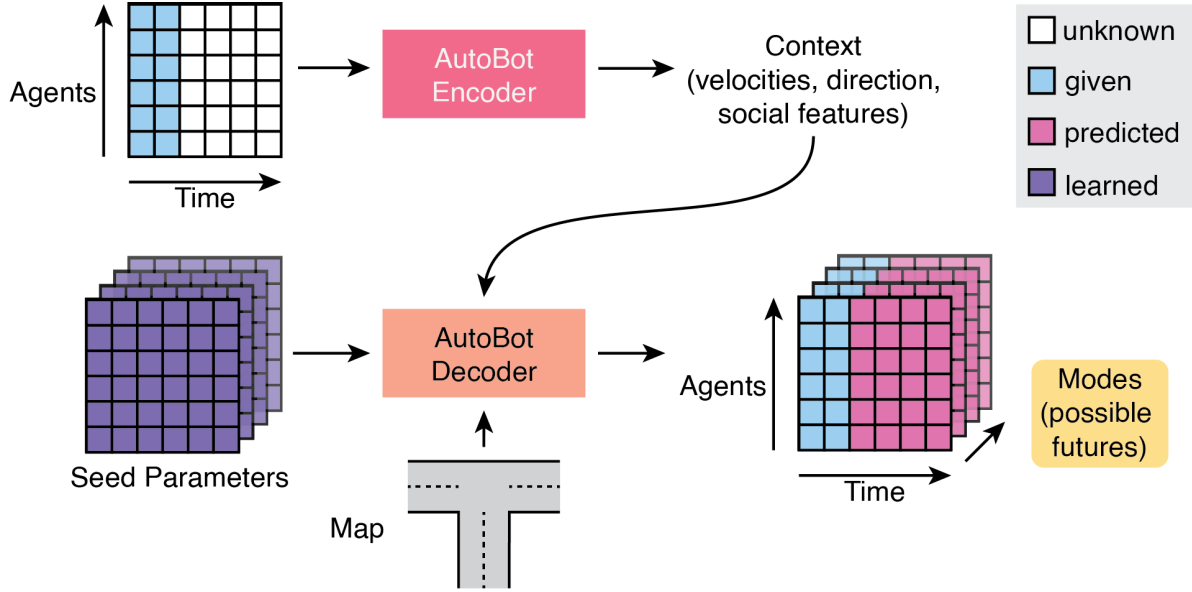


Figure 4.1 **Overview:** Our model generates multiple possible futures for agents based on several input timesteps. The input trajectories are encoded into a context tensor that captures their respective behavior and interaction with surrounding agents. While Transformer-based architectures are often autoregressive, we decode all future steps at once. This is accomplished via learnable seed parameters. Our decoder transforms these, together with the map and the context into the possible future trajectories that we call “modes”.

4.4 Latent Variable Sequential Set Transformers

Latent Variable Sequential Set Transformers is a class of encoder-decoder architectures that process sequences of sets (see Fig 4.2). In this section, we describe the encoding and decoding procedures, the latent variable mechanism, and the training objective. Additionally, we provide proof of the permutation equivariance of our model in Appendix A.1, and implementation details in Appendix A.2 as well as a discussion of the special case of AutoBot-Ego, which is similar to AutoBots but predicts a future sequence for only one element in a scene.

4.4.1 Encoder: Input Sequence Representation

AutoBot takes as an input a sequence of sets, $\mathbf{X}_{1:t} = (\mathbb{X}_1, \dots, \mathbb{X}_t)$, which in the motion prediction setting may be viewed as the state of a scene evolving over t timesteps. Let us describe each set as having M elements (or agents) with K attributes (e.g. x/y position). To process the social and temporal information the encoder applies the following two transformations.

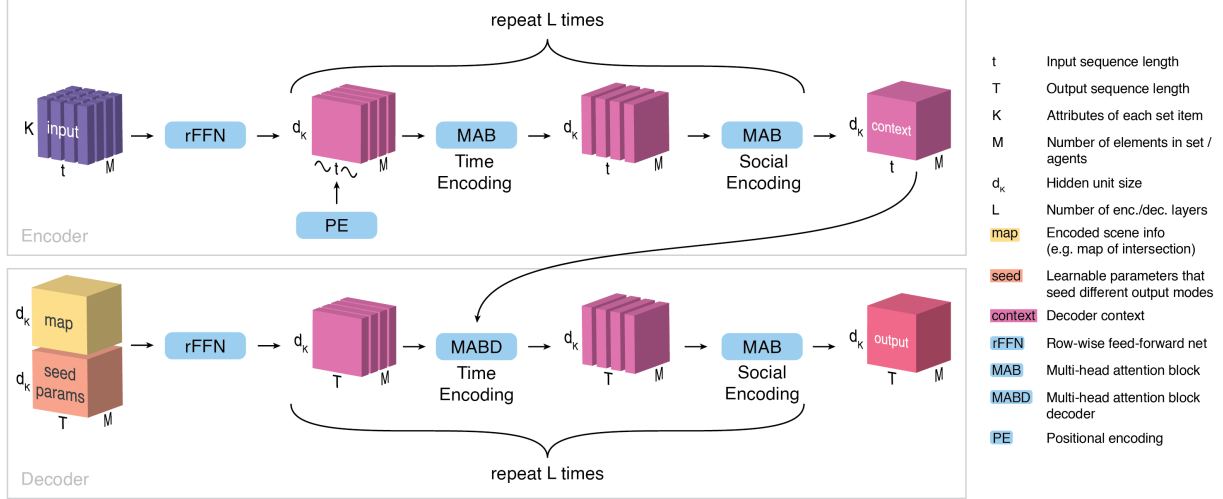


Figure 4.2 Architecture Overview. Our model takes as input a tensor of dimension K, M, t . A row-wise feed-forward network (rFFN) is applied to each row along the $t \times M$ plane transforming vectors of dimension K to d_K . After adding positional encoding (PE) to the t axis, the encoder passes the tensor through L repeated layers of multi-head attention blocks (MAB) that are applied to the time axis (time encoding) and the agent axis (social encoding) before outputting the context tensor. In the decoder, the encoded map and the learnable seed parameters tensor are concatenated and passed through an rFFN before being passed through L repeated layers of a multi-head attention block decoder (MABD) along the time axis (using the context from the encoder) followed by a MAB along the agent axis. This figure shows the process for predicting one possible future trajectory (“mode”). When multiple modes are predicted, each mode has its own learnable seed parameters and the decoder is rolled out once for each mode.

Appendix A.1.3 provides a proof of equivariance with respect to the agent dimension for this encoder.

First, the AutoBots encoder injects temporal information into the sequence of sets using a sinusoidal positional encoding function $\text{PE}(\cdot)$ [48]. For this step, we view the data as a collection of matrices, $\{\mathbf{X}_0, \dots, \mathbf{X}_M\}$, that describe the temporal evolution of agents. The encoder processes temporal relationships between sets using a MAB, performing the following operation on each set where $m \in \{1, \dots, M\}$:

$$\mathbf{S}_m = \text{MAB}(\text{rFFN}(\mathbf{X}_m))$$

where $\text{rFFN}(\cdot)$ is an embedding layer projecting each element in the set to the size of the hidden state d_K . Next, we process temporal slices of $\mathbf{S}$, retrieving sets of agent states $\mathbf{S}_\tau$ at

some timestep τ and is processed using $\mathbb{S}_\tau = \text{MAB}(\mathbb{S}_\tau)$. These two operations are repeated L_{enc} times to produce a **context tensor** $\mathbf{C} \in \mathbb{R}^{(d_K, M, t)}$ summarizing the entire input scene, where t is the number of timesteps in the input scene.

4.4.2 Decoder: Multimodal Sequence Generation

The goal of the decoder is to generate temporally and socially consistent predictions in the context of multi-modal data distributions. To generate $c \in \mathbb{N}$ different predictions for the same input scene, the AutoBot decoder employs c matrices of learnable seed parameters $\mathbf{Q}_i \in \mathbb{R}^{(d_K, T)}$ where T is the prediction horizon, where $i \in \{1, \dots, c\}$. Intuitively, each matrix of learnable seed parameters corresponds to a setting of the discrete latent variable in AutoBot. Each learnable matrix $\mathbf{Q}_i$ is then repeated across the agent dimension M times to produce the input tensor $\mathbf{Q}'_i \in \mathbb{R}^{(d_K, M, T)}$. Additional contextual information (such as a rasterized image of the environment) is encoded using a convolutional neural network to produce a vector of features $\mathbf{m}_i$ where $i \in \{1, \dots, c\}$. In order to provide the contextual information to all future timesteps and to all set elements, we copy this vector along the M and T dimensions, producing the tensor $\mathbf{M}_i \in \mathbb{R}^{(d_K, M, T)}$. Each tensor $\mathbf{Q}'_i$ is concatenated with $\mathbf{M}_i$ along the d_K dimension, as shown on the left side of Figure 4.2 (bottom). This tensor is then processed using a rFFN(.) to produce the tensor $\mathbf{H} \in \mathbb{R}^{(d_K, M, T)}$.

We begin to decode by processing the time dimension, conditioning on the encoder’s output $\mathbf{C}$, and the encoded seed parameters and environmental information in $\mathbf{H}$. The decoder processes each of m agents in $\mathbf{H}$ separately with an MABD layer:

$$\mathbf{H}'_m = \text{MABD}(\mathbf{H}_m, \mathbf{C}_m)$$

where $\mathbf{H}'$ is the output tensor that encodes the future time evolution of each element in the set independently. In order to ensure that the future scene is socially consistent between set elements, we process each temporal slice of $\mathbf{H}'$, retrieving sets of agent states $\mathbb{H}'_\tau$ at some future timestep τ where each element $\mathbf{h}'_\tau \in \mathbb{H}'_\tau$ is processed by a MAB layer. Essentially, the MAB layer here performs a per-timestep attention between all set elements, similar to the Set Transformer [49].

These two operations are repeated L_{dec} times to produce a final output tensor for mode i . The decoding process is repeated c times with different learnable seed parameters $\mathbf{Q}_i$ and additional contextual information $\mathbf{m}_i$. The output of the decoder is a tensor $\mathbf{O} \in \mathbb{R}^{(d_K, M, T, c)}$ which can then be processed element-wise using a neural network $\phi(\cdot)$ to produce the desired output representation. In several of our experiments, we generate trajectories in (x, y) space,

and as such, ϕ produces the parameters of a bivariate Gaussian distribution.

Note on the learnable seed parameters. One of the main contributions that make AutoBot’s inference and training time faster than prior work is the use of the decoder seed parameters $\mathbf{Q}$. These parameters serve a dual purpose. Firstly, they account for the diversity in future prediction, where each matrix $\mathbf{Q}_i \in \mathbb{R}^{(d_k, T)}$ (where $i \in \{1, \dots, c\}$) corresponds to one setting of the discrete latent variable. Secondly, they contribute to the speed of AutoBot by allowing it to perform inference on the entire scene with a single forward pass through the decoder without sequential sampling. Prior work (e.g., [194]) uses auto-regressive sampling of the future scene with interlaced social attention, which allows for socially consistent multi-agent predictions. Other work (e.g., [195]) employ a single output MLP to generate the entire future of each agent *independently* with a single forward pass. Using the attention mechanisms of transformers, AutoBot combines the advantages of both approaches by deterministically decoding the future scene starting from each seed parameter matrix. In Appendix A.2.5, we compare the inference speed of AutoBots with their autoregressive counterparts.

4.4.3 Training Objective

Given a dataset $\mathcal{D} = \{(\mathbb{X}_1, \dots, \mathbb{X}_T)\}_{i=1}^N$ consisting of N sequences, each having T sets, our goal is to maximize the likelihood of the future trajectory of all elements in the set given the input sequence of sets, i.e., $\max p(\mathbf{X}_{t+1:T} | \mathbf{X}_{1:t})$. In order to simplify notation, we henceforth refer to the ego-agent’s future trajectory by $\mathcal{Y} = \mathbf{X}_{t+1:T}$. As discussed above, AutoBot employs discrete latent variables, which allows us to compute the log-likelihood exactly. The gradient of our objective function can then be expressed as follows:

$$\begin{aligned} \nabla_{\theta} \mathcal{L}(\theta) &= \nabla_{\theta} \log p_{\theta}(\mathcal{Y} | \mathbf{X}_{1:t}) = \nabla_{\theta} \log \left(\sum_Z p_{\theta}(\mathcal{Y}, Z | \mathbf{X}_{1:t}) \right) \\ &= \sum_Z p_{\theta}(Z | \mathcal{Y}, \mathbf{X}_{1:t}) \nabla_{\theta} \log p_{\theta}(\mathcal{Y}, Z | \mathbf{X}_{1:t}) \end{aligned} \quad (4.3)$$

As previously discussed in [194], computing the posterior likelihood $p_{\theta}(Z | \mathcal{Y}, \mathbf{X}_{1:t})$ is difficult in general and varies with θ . As discussed in [31], one can introduce a distribution over the latent variables, $q(Z)$, that is convenient to compute. With this distribution, we can now decompose the log-likelihood as follows:

$$\log p_{\theta}(\mathcal{Y} | \mathbf{X}_{1:t}) = \sum_Z q(Z) \log \frac{p_{\theta}(\mathcal{Y}, Z | \mathbf{X}_{1:t})}{q(Z)} + D_{KL}(q(Z) || p_{\theta}(Z | \mathcal{Y}, \mathbf{X}_{1:t})), \quad (4.4)$$

where $D_{KL}(\cdot||\cdot)$ is the Kullback-Leibler divergence between the approximating posterior and the actual posterior. A natural choice for this approximating distribution is $q(Z) = p_{\theta_{old}}(Z|\mathcal{Y}, \mathbf{X}_{1:t})$, where θ_{old} corresponds to AutoBot’s parameters before performing the parameter update. With this choice of $q(Z)$, the objective function can be re-written as

$$\begin{aligned}\mathcal{L}(\theta) &= Q(\theta, \theta_{old}) + \text{const.} \\ Q(\theta, \theta_{old}) &= \sum_Z p_{\theta_{old}}(Z|\mathcal{Y}, \mathbf{X}_{1:t}) \log p_{\theta}(\mathcal{Y}, Z|\mathbf{X}_{1:t}) \\ &= \sum_Z p_{\theta_{old}}(Z|\mathcal{Y}, \mathbf{X}_{1:t}) \left\{ \log p_{\theta}(\mathcal{Y}|Z, \mathbf{X}_{1:t}) + \log p_{\theta}(Z|\mathbf{X}_{1:t}) \right\}.\end{aligned}\tag{4.5}$$

where the computation of $p_{\theta}(Z|\mathbf{X}_{1:t})$ is described in Appendix A.2.2. Note that the posterior $p_{\theta_{old}}(Z|\mathcal{Y}, \mathbf{X}_{1:t})$ can be computed exactly in the model we explore here since AutoBot operates with discrete latent variables. Therefore, our final objective function becomes to maximize $Q(\theta, \theta_{old})$ and minimize $D_{KL}(p_{\theta_{old}}(Z|\mathcal{Y}, \mathbf{X}_{1:t})||p_{\theta}(Z|\mathbf{X}_{1:t}))$.

As mentioned, for our experiments we use a function ϕ to produce a bivariate Gaussian distribution for each agent at every timestep. We found that to ensure that each mode’s output sequence does not have high variance, we introduce a mode entropy (ME) regularization term to penalize large entropy values of the output distributions,

$$L_{ME} = \lambda_e \max_Z \sum_{\tau=t+1}^T H(p_{\theta}(\mathbb{X}_{\tau}|Z, \mathbf{X}_{1:t})).\tag{4.6}$$

As we can see, this entropy regularizer penalizes only the mode with the maximum entropy. Our results in Section A.3.2 show the importance of this term in learning to cover the modes of the data.

4.5 Experiments

We now demonstrate the capability of AutoBots to model sequences of set-structured data across several domains. In subsection 4.5.1 we show strong performance on the competitive motion-prediction dataset nuScenes [190], a dataset recorded from a self-driving car, capturing other vehicles and the intersection geometry. In subsection 4.5.2, we demonstrate our model on the Argoverse autonomous driving challenge. In subsection 4.5.3, we show results on the synthetic partition of the TrajNet++ [189] dataset which contains multi-agent scenes with a high degree of interaction between agents. Finally in subsection 4.5.4, we show that our model can be applied to generate characters conditioning across strokes, and verify that AutoBot-Ego is permutation-invariant across an input set. See Appendix A.3.2, for

an illustrative toy experiment that highlights our model’s ability to capture discrete latent modes.

4.5.1 NuScenes, A Real-World Driving Dataset

In this section, we present AutoBot-Ego’s results on the nuScenes dataset. The goal of this benchmark dataset is to predict the ego-agent’s future trajectory (6 seconds at 2hz) given the past trajectory (up to 2 seconds at 2hz). Additionally, the benchmark provides access to all neighbouring agents’ past trajectories and a birds-eye-view RGB image of the road network in the vicinity of the ego-agent. Figure 4.3 shows three example predictions produced by AutoBot-Ego trained with $c = 10$. We observe that the model learns to produce trajectories that are in agreement with the road network and covers most possible futures directions. We also note that for each direction, AutoBot-Ego assigns different modes to different speeds to more efficiently cover the possibilities.

We compare our results on this benchmark to the state-of-the-art in Table 4.1. **Min ADE (5)** and **(10)** are the average of pointwise L2 distances between the predicted trajectory and ground truth over 5 and 10 most likely predictions respectively. A prediction is classified as a *miss* if the maximum pointwise L2 distance between the prediction and ground truth is greater than 2 meters. **Miss Rate Top-5 (2m)** and **Top-10 (2m)** are the proportion of misses over all agents, where for each agent, 5 and 10 most likely predictions respectively are evaluated to check if they’re misses. **Min FDE (1)** is the L2 distance between the final points of the prediction and ground truth of the most likely prediction averaged over all agents. **Off Road Rate** is the fraction of predicted trajectories that are not entirely contained in the drivable area of the map.

We can see that AutoBot-Ego obtains the best performance on the Min ADE (10) metric and on Off Road Rate metric, and strong performance on the other metrics. Furthermore, using an ensemble of three AutoBot-Ego models slightly improves the performance. This demonstrates our model’s ability to capture multi-modal distributions over future trajectories due to the latent variables. AutoBot-Ego excelled in maintaining predictions on the road, showing that the model has effectively used its capacity to attend to the map information. Finally, we note that like all other models, AutoBot-Ego struggles to predict the probability of the correct mode, which results in poor performance of the Min FDE (1). The most interesting part of AutoBot-Ego is its computational requirements. We highlight that AutoBot-Ego was trained on a single Nvidia GTX 1080 Ti for 3 hours. This is in large part due to the use of the decoder seed parameters which allow us to predict the future trajectory of the ego-agent in a single forward pass. Note that in Appendix A.2.5, we present the inference speed of AutoBot-Ego

Model	Min ADE (5)	Min ADE (10)	Miss Rate Top-5 (2m)	Miss Rate Top-10 (2m)	Min FDE (1)	Off Road Rate
Noah_prediction	1.59	1.37	0.69	0.62	9.23	0.08
CXX	1.63	1.29	0.69	0.60	8.86	0.08
LISA(MHA_JAM)	1.81	1.24	0.59	0.46	8.57	0.07
Trajectron++	1.88	1.51	0.70	0.57	9.52	0.25
CoverNet	2.62	1.92	0.76	0.64	11.36	0.13
Physics Oracle	3.70	3.70	0.88	0.88	9.09	0.12
WIMP	1.84	1.11	0.55	0.43	8.49	0.04
GOHOME	1.42	1.15	0.57	0.47	6.99	0.04
AutoBot-Ego ($c=10$)	1.43	1.05	0.66	0.45	8.66	0.03
AutoBot-Ego (ensemble)	1.37	1.03	0.62	0.44	8.19	0.02

Table 4.1 **Quantitative Results on the nuScenes dataset.** Other methods: LISA [195]; Trajectron++ [196]; CoverNet [197]; Physics Oracle [190]; WIMP [198]; GOHOME [199]

compared to an auto-regressive version. Ablation studies on various components of AutoBot-Ego are also presented in Appendix A.3.1.

4.5.2 Argoverse Results

Table 4.2 shows the results of AutoBot-Ego (with $c = 6$) on the Argoverse test set. While our method does not achieve top scores, it uses much less computation, less data augmentation and no ensembling compared to alternatives. Further, we compare quite well to the 2020 Argoverse competition winner, a model called “Jean”. In addition, we note that the Argoverse challenge does not restrict the use of additional training data and that many of the top entries on the leaderboard have no publicly available papers or codebases. Many methods are therefore optimized for Argoverse by means of various data augmentations. For example, Scene Transformer [16] used a data augmentation scheme that includes agent dropout and scene rotation which we have not, and the current top method and winner of the 2021 challenge (QCraft) makes use of ensembling.

We believe that a fair comparison of the performance between neural network methods should ensure that each architecture is parameter-matched and the training is FLOP-matched. While the Argoverse leaderboard does not require entrants to report their parameter or FLOP count, we report available public statistics to enable comparison here. First, Scene Transformer [16] reports that their model trains in 3 days on TPUs and TPU-v3s have a peak performance of 420 TeraFLOPS, which means their model could have consumed up to 3 days $\times$ 24 hours $\times$ 420 TFLOPS, or 108,000,000 TFLOPS. On the other hand, AutoBot-Ego was trained on a Nvidia 1080ti for 10 hours, consuming roughly 396,000 TFLOPs, or 0.4% as much. Jean’s method [200] consumes roughly 8,600,000 TFLOPs (training our method uses

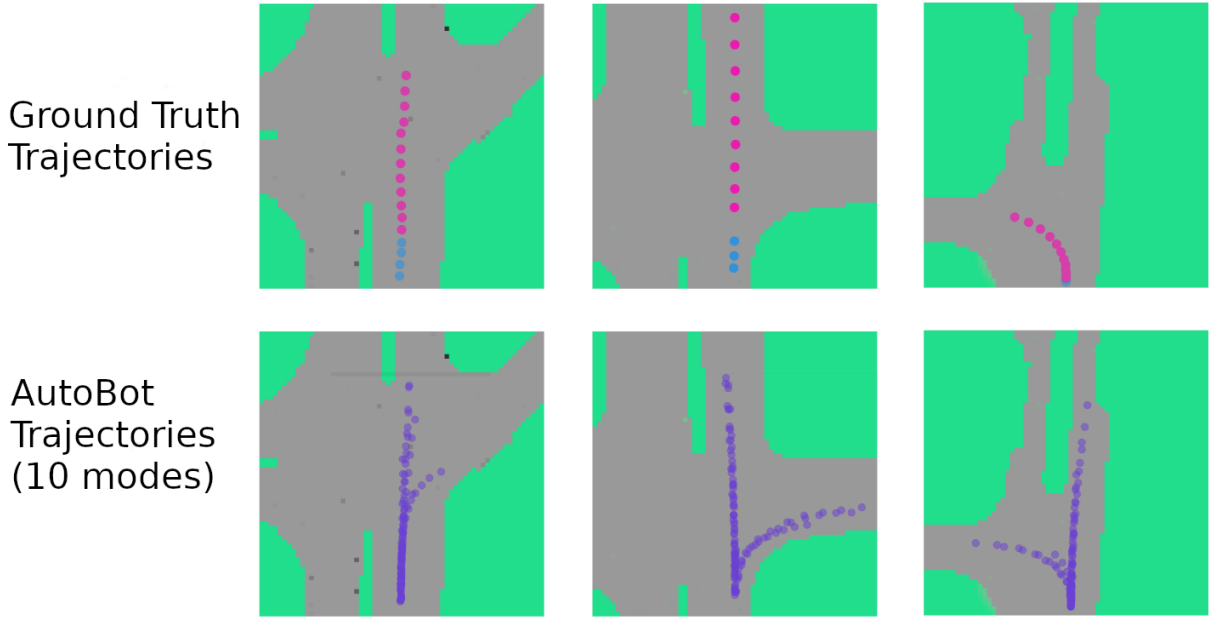


Figure 4.3 **NuScenes Trajectory Prediction**. Top row: birds-eye-view of road network with ground truth trajectory data where given trajectory is cyan and held-out trajectory information is pink. Bottom row: diverse trajectories generated by the different modes of AutoBot-Ego. The model generates trajectories that adhere to the road network and captures distinct possible futures.

only 5% of that compute). While the method currently topping the leaderboard has not revealed their model details, the fact that they used ensembles very likely requires significantly more compute than our method.

4.5.3 TrajNet++ Synthetic Dataset

In this section, we demonstrate the utility of the having social and temporal attention in the encoder and decoder of AutoBot. We chose to evaluate on the synthetic partition of the TrajNet++ dataset since this was specifically designed to have a high-level of interaction between scene agents [192]. In this task, we are provided with the state of all agents for the past 9 timesteps and are tasked with predicting the next 12 timesteps for all agents. The scene is initially normalized around one (ego) agent such that the agent’s final position in the input sequence is at (0,0) and its heading is aligned with the positive y direction. There are a total of 54,513 unique scenes in the dataset, which we split into 49,062 training scenes and 5,451 validation scenes. For this experiment, we apply our general architecture

Model	Min ADE ($\downarrow$)	Min FDE ($\downarrow$)	Miss Rate ($\downarrow$)	DAC ($\uparrow$)
AutoBot-Ego (Valid Set)	0.73	1.10	0.12	-
AutoBot-Ego (Test Set)	0.89 (top-5)	1.41 (top-5)	0.16 (top-6)	0.9886 (top-3)
Jean (2020 winner)	0.97	1.42	0.13	0.9868
WIMP	0.90	1.42	0.17	0.9815
TNT	0.94	1.54	0.13	0.9889
LaneGCN	0.87	1.36	0.16	0.9800
TPCN	0.85	1.35	0.16	0.9884
mmTransformer	0.84	1.34	0.15	0.9842
GOHOME	0.94	1.45	0.105	0.9811
Scene Transformer	0.80	1.23	0.13	0.9899

Table 4.2 **Quantitative Results on the Argoverse test set.** Other methods that had an associated paper: Jean [200]; WIMP [198]; TNT [201]; LaneGCN [202]; TPCN [203]; mmTransformer [204]; GOHOME [199]; Scene Transformer [16].

(AutoBot) to multi-agent scene forecasting, and perform an ablation study on the social attention components in the encoder and the decoder.

Table 4.3 presents three variants of the model, and a baseline extrapolation model which shows that the prediction problem cannot be trivially solved. The first, AutoBot-AntiSocial, corresponds to a model without the intra-set attention functions in the encoder and decoder (see Figure 4.2). The second variant, AutoBot-Ego, only sees the past social context and has no social attention during the trajectory generation process. The third model corresponds to the general architecture, AutoBot. All models were trained with $c = 6$ and we report scene-level error metrics as defined by [14]. Using the social attention in the decoder significantly reduces the number of collisions between predicted agents’ futures and significantly

Model	Ego Agent’s Min ADE(6) ($\downarrow$)	Number Of Collisions ($\downarrow$)	Scene-level Min ADE(6) ($\downarrow$)	Scene-level Min FDE(6) ($\downarrow$)
Linear Extrapolation	0.439	2220	0.409	0.897
AutoBot-AntiSocial	0.196	1827	0.316	0.632
AutoBot-Ego	0.098	1144	0.214	0.431
AutoBot	0.095	139	0.128	0.234

Table 4.3 **TrajNet++ ablation studies for a multi-agent forecasting scenario.** We investigate the impact of using the social attention in the encoder and in the decoder. We found that AutoBot is able to cause significantly fewer collisions between agents compared to variants without the social attention, and improves the prediction accuracy on the scene-level metrics while maintaining strong performance on the ego-agent. Metrics are reported on the validation set.

outperforms its ablated counterparts on scene-level minADE and minFDE. We expect this at a theoretical level since the ego-centric formulation makes the independence assumption that the scene’s future rollout can be decomposed as the product of the individual agent’s future motion in isolation, as remarked in [14]. AutoBot does not make this assumption as it forecasts the future of all agents jointly. We refer the reader to Appendix A.3.3 for additional ablation experiments and qualitative results on this dataset.

4.5.4 Omniglot, Stroke-based Image Generation

To demonstrate the general efficacy of our approach to sequence generation, we demonstrate AutoBots on a character completion task, a diverse alternative to automotive and pedestrian trajectory prediction. We set up two different tasks on the Omniglot dataset [193]: (a) **stroke-completion task** in which the model is presented with the first half of each stroke and has to complete all of them in parallel and (b) **character-completion task** in which the model is given several full strokes and has to draw the last stroke of the character. In task (a), the model is trained on the entire Omniglot dataset (training set), in which strokes are encoded as sequential sets of points, and we are comparing the performance of AutoBot qualitatively against that of an LSTM baseline with social attention and discrete latents (similar to AutoBot but with classic RNN architecture) on a held-out test set. With the first task, we illustrate that our model is flexible enough to learn to legibly complete characters from a variety of different languages. In task (b), we train AutoBot-Ego only on characters “F”, “H”, and “II” to demonstrate that our discrete latent variable captures plausible character completions given the context of the other strokes, e.g. a character “F” can be completed as an “H” when the top stroke is missing. The results of these experiments can be found in Fig. 4.4. Implementation details and additional results can be found in Appendix section A.3.5.

4.6 Related Work

Processing set structured data. Models of set-valued data (i.e. unordered collections) should be permutation-invariant (see Appendix A.1 for formal definitions), and be capable of processing sets with arbitrary cardinality. Canonical forms of feed forward neural networks and recurrent neural network models do not have these properties [49]. One type of proposed solution is to integrate pooling mechanisms with feed forward networks to process set data [205, 206, 207]. This is referred as social pooling in the context of motion prediction [11, 12, 13]. Attention-based models [14, 15, 16, 17, 49] or graph-based models [103, 195, 201, 208] have performed well in set processing tasks that require modelling

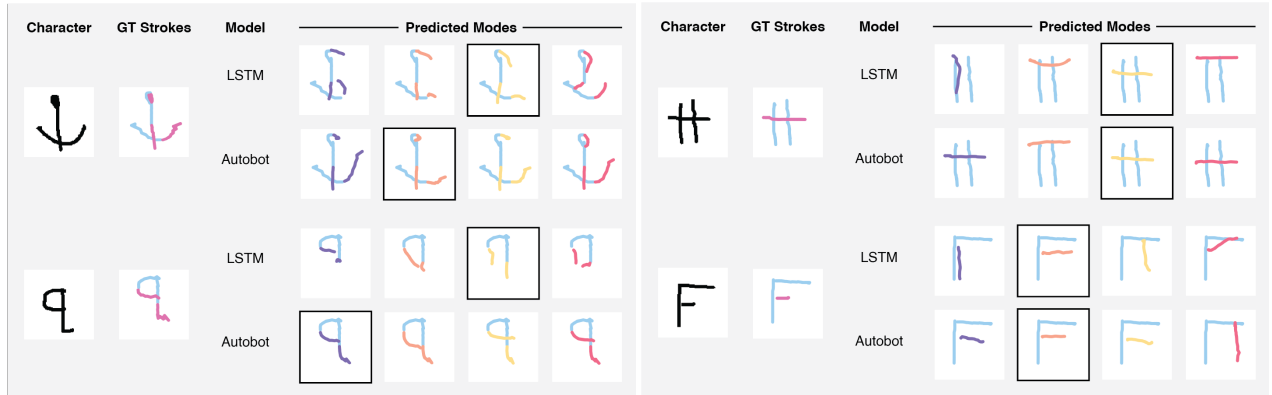


Figure 4.4 **Omniglot qualitative results.** *Left: stroke completion task.* We show examples of characters generated using AutoBot and an LSTM baseline. The first two columns show the ground-truth image of the character and the corresponding ground-truth strokes. In this task, the models are provided with the first half of all strokes (cyan) and are tasked with generating the other half (pink). The four other columns show the generated future strokes, one for each latent mode. We can see that AutoBot produces more consistent and realistic characters compared to the LSTM baseline. *Right: character completion task.* We show two examples of characters completed using AutoBot-Ego and an LSTM baseline. The first two columns show the ground-truth image of the character as before. In this case, the models are provided with two complete strokes and are tasked with generating a new stroke to complete the character. We observe that AutoBot-Ego generates more realistic characters across all modes, given ambiguous input strokes.

high-order element interactions. In contrast, we provide a joint single pass attention-based approach for modelling more complex data-structures composed by sets **and** sequence, while also addressing how to generate diverse and heterogeneous data-structures. Although those properties exist in concurrent motion prediction papers [14, 15, 16, 194, 209], our proposed method is targeting a more general set of problems while still getting competitive results in motion prediction benchmarks at a fraction of other models’ compute.

Diverse set generation. Generation of diverse samples using neural sequence models is an important part of our model and a well-studied problem. For example, in [210] a variational (CVAE) approach is used to construct models that condition on attributes for dialog generation. The CVAE approach has also been integrated with transformer networks [211, 212] for diverse response generation. For motion prediction, [13] and [12] use a variational mechanism where each agent has their own intentions encoded. [209] output Gaussian mixtures at every timestep and sample from it to generate multiple futures. The most similar strategy to ours was developed by [194]. Other recent works [14, 34] use continuous latent variables to model the multiple actors’ intentions jointly. We built on this method by using one joint transformer conditioned on the latent variables and added a maximum entropy likelihood loss that improves the variability of the generated outcomes.

Sequential state modelling. We are interested in data that can be better represented as a sequence. The main example of this is language modelling [48, 213], but also includes sound [214], video [215], and others [216, 217]. For motion prediction, prior work has largely focused on employing RNNs to model the input and/or output sequence [11, 12, 13, 14, 194, 195, 208, 209]. With its recent success in natural language processing, Transformers [48] have been adopted by recent approaches for motion forecasting.[218] uses a graph encoding strategy for pedestrian motion prediction. [219] and [204] propose a transformer-based encoding to predict the future trajectories of agents separately. In contrast, AutoBots produces representations that jointly encode both the time and social dimensions by treating this structure as a sequence of sets. This provides reuse of the model capacity and a lightweight approach. Concurrent to our method, [15], also jointly encodes social and time dimensions but flattens both of those axis into a single dimension.

4.7 Conclusion

In this paper, we propose the Latent Variable Sequential Set Transformer to model time-evolution of sequential sets using discrete latent variables. We make use of the multi-head attention block to efficiently perform intra-set attention, to model the time-dependence between the input and output sequence, and to predict the prior probability distribution over

the latent modes. We validate our AutoBot-Ego by achieving competitive results on the nuScenes benchmark for ego-centric motion forecasting in the domain of autonomous vehicles. We further demonstrate that AutoBot can model diverse sequences of sets that adhere to social conventions. We validate that since our model can attend to the hidden state of all agents during the generative process, it produces scene-consistent predictions on the TrajNet++ dataset when compared with ego-centric forecasting methods.

Limitations. Our approach is limited to operate in a setting where a perceptual pipeline is providing structured sets representing the other entities under consideration. If our approach is used in a robot or autonomous car, any errors or unknown entities not captured by sets used as input to the algorithm would not be modelled by our approach. Memory limitations may also limit the number of sets that can be processed at any time. In future work, we plan to extend this line of work to broader and more interpretable scene-consistency. Further, we would like to investigate the combination of a powerful sequence model like AutoBot with a reinforcement learning algorithm to create a data-efficient model-based RL agent for multi-agent settings.

CHAPTER 5 CONTROLLABLE DRIVING BEHAVIOUR SIMULATION WITH OFFLINE REINFORCEMENT LEARNING

In the previous chapter, we showed how latent variable sequential set transformers can effectively model multi-agent interactions, and can generate diverse and realistic future scenes. In this chapter, we examine how real-world driving data can be leveraged to evaluate an autonomous-vehicle (AV) planning stack, contrasting this with the classical simulation approach of replaying recorded logs. The latter results in unrealistic simulations that have non-reactive agents or agents that are heuristically controlled using handcrafted rules. More recent approaches employ generative models from real-world data but these may necessitate costly iterative sampling to control and cannot effectively sample long-tail simulations. In this chapter, we focus our attention on methods to better simulate other agents’ behaviour with the goal of controllability and reactivity. To that end, we propose CtRL-Sim, a method that leverages return-conditioned offline reinforcement learning to generate traffic agents’ behaviours. We start with real-world driving data that is processed through a physics-enhanced Nocturne simulator to ensure the realism of state transitions. This results in an offline RL dataset with multi-component reward annotations. Next, we train a return-conditioned multi-agent behaviour model with a specific factorization of the joint distribution of actions, returns and state-rollouts such that we can sample diverse and interesting behaviours at inference time. We show that CtRL-Sim can generate realistic safety-critical scenarios while providing fine-grained control over agent behaviours. The contributions of this chapter are:

1. We propose CtRL-Sim, which is to the best of our knowledge, the first mathematical framework applying return-conditioned offline RL for controllable and reactive behaviour simulation. Specifically, CtRL-Sim uses exponential tilting of the factorized return-to-go to control different axes of agent behaviour.
2. To enable multi-agent return-conditioned policy learning, we propose an autoregressive multi-agent encoder-decoder Transformer architecture that models reactive interactions among agents while conditioning on the factorized return-to-go.
3. We extend the Nocturne simulator [38] with a Box2D physics engine, which facilitates realistic vehicle dynamics and collision interactions.

5.1 Introduction

The planning stacks of autonomous vehicles have benefited greatly from recent research in safe navigation of urban environments [220]. However, this progress is still limited to nominal driving environments, where the behaviours of pedestrians and other drivers are expected. With the goal of widespread adoption, it is imperative that we train and/or evaluate our driving policies on long-tail scenarios. While simulation can be efficiently used to validate the safety of a driving stack on long-tail scenarios, a core challenge with such simulators is their ability to generate realistic and diverse driving behaviors of other agents while ensuring their reactivity to the ego-agent’s (and other agents’) actions. An important aspect of a simulator is agent controllability as it would allow designers to scale the evaluation efficiently and ensure robustness to a spectrum of other agent behaviors.

One common solution to simulation is to fix other agents behaviours to those exhibited in driving logs. However, this is clearly lacking the reactivity requirement, yielding unrealistic interactions, and therefore, uninformative to the planning stack designers. To address reactivity, other approaches have proposed rule-based policies [32, 33]. However, rule-based policies will often lack diversity, and can be largely unrealistic due to heuristic rules. Generative models learned from real-world data have come to center-stage as they can greatly enhance the realism of simulated agent behaviours[27, 34, 35, 36, 125, 130]. However, these approaches still lack controllability [27, 34, 35] or require costly sampling procedures for behavior control [36, 128, 130, 132, 133].

In this chapter, we propose CtRL-Sim to address the limitations of prior work. CtRL-Sim utilizes return-conditioned offline reinforcement learning (RL) to enable reactive, *closed-loop*, controllable, and probabilistic behaviour simulation within a physics-enhanced Nocturne [38] environment. Our approach relies on using the physics-enhanced Nocturne environment to replay scenarios from the Waymo Open Motion Dataset to curate an offline RL dataset. We annotate this dataset with different reward terms including off-road rewards, vehicle-to-vehicle collisions and reaching a goal. We then propose a return-conditioned multi-agent autoregressive Transformer architecture [140] within the CtRL-Sim framework to imitate the driving behaviours in the curated dataset. Since CtRL-Sim is fully probabilistic, we leverage exponential tilting [40] as a mechanism to produce a spectrum of behaviors given identical initial conditions. While Lee et al.[40] proposed this tilting in the realm of reward maximization, we explore the full space including reward minimization to produce adversarial, long-tail, behaviors. Figure 5.1 shows an overview of how the tilting mechanism of CtRL-Sim is used to generate diverse behavior. As we depict in the figure, the controllable generation relies on the use of a tilting coefficient for each reward component, circumventing any costly

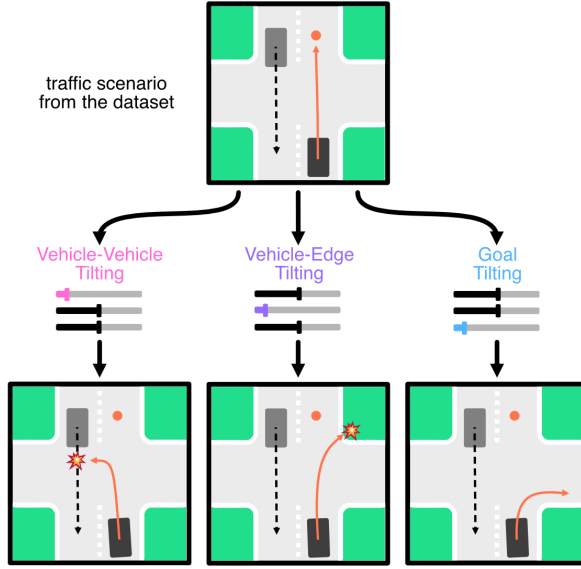


Figure 5.1 **CtRL-Sim Overview** [26]. CtRL-Sim allows for controllable agent behaviour from existing datasets. This allows users to create interesting edge cases for testing and evaluating AV planners.

iterative sampling approaches.

We demonstrate the effectiveness of CtRL-Sim at producing controllable and realistic agent behaviors compared to prior methods. Furthermore, we show that finetuning the base models on adversarial scenarios further enhances the controllability of adversarial behaviors while maintaining realism. We believe that CtRL-Sim can serve as a useful framework for enhancing the safety and robustness of AV planners by using the simulator for training and evaluation.

5.2 CtRL-Sim

In this section, we present the CtRL-Sim framework for behaviour simulation. We start by introducing CtRL-Sim in the single-agent setting, and subsequently show its extension to the multi-agent setting. Given the state of an agent s_t at timestep t and additional context (e.g., the road structure, the agent’s goal), the behaviour simulation model employs a driving policy $\pi(a_t|s_t, m, s_G)$ and a forward transition model $\mathcal{P}(s_{t+1}|s_t, a_t)$ to control the agent through the scene. Note that a_t is the action, m is the map context, and s_G is the prescribed goal state. In this work, we employ a physics-enhanced Nocturne simulator, providing a physically-realistic forward transition model $\mathcal{P}$. All we are missing is modeling the policy $\pi(a_t|s_t, m, s_G)$. The desiderata for the policy, π , are that it should induce a distribution over trajectories that

matches real driving behaviour and admits interpretable control over an agent’s behaviour, enabling the generation of long-tail counterfactual scenarios.

5.2.1 Controllable Simulation via Offline RL

We consider the common offline RL setup where we are given a dataset $\mathcal{D}$ of trajectories $\tau_i = \{\dots, s_t, a_t, r_t, \dots\}$, with states $s_t \in \mathcal{S}$, actions $a_t \in \mathcal{A}$ and rewards r_t . A potentially unknown and possibly suboptimal behaviour policy, $\pi_B(a_t|s_t)$, is responsible for generating this dataset of trajectories through its execution in a finite-horizon Markov decision process. At any timestep t , we define the *return-to-go* as the cumulative sum of rewards collected from that point until the end of the trajectory:

$$G_t = \sum_{t'=t}^T r_{t'}. \quad (5.1)$$

The central goal of offline reinforcement learning is to learn a policy π that achieves performance comparable to, or exceeding, the best behaviors demonstrated in the dataset $\mathcal{D}$.

The primary insight of this work is the observation that offline RL can be an effective way to perform controllable simulation. Specifically, at inference time, we can tilt the distribution over actions towards specific behaviours by adjusting the values of the return-to-go G_t . This requires a different formulation of the policy such that it is conditioned on the return $\pi(a_t|s_t, G_t, s_G)$ ¹. In Table 3.1, we present the different approaches that have been explored in the Offline RL literature focusing on learning return-conditioned policies. Our approach in this work is to model the joint distribution over returns and actions of an agent for a given dataset. Specifically, we adopt a specific factorization which ensures that the policy is conditioned on the return-to-go G_t ,

$$p_\theta(a_t, G_t|s_t, s_G) = \pi_\theta(a_t|s_t, s_G, G_t)p_\theta(G_t|s_t, s_G) \quad (5.2)$$

We note that [139] found it helpful to also utilize a *model-based* return-conditioned policy, whereby the future state is modelled as part of the joint distribution being learned. Gontier et al. [139] showed that this objective provides a useful regularizing signal for the policy, even though the future state prediction is not directly used at inference time. In this work, we extend this approach to predicting the full future state rollout of the agent; this also

¹Note that we omit the additional context m for brevity.

provided a regularizing effect. The final distribution CtrL-Sim models is thus given by

$$p_\theta(s_{t+1:T}, a_t, G_t | s_t, s_G) = p_\theta(s_{t+1:T} | s_t, s_G, G_t, a_t) \pi_\theta(a_t | s_t, s_G, G_t) p_\theta(G_t | s_t, s_G) \quad (5.3)$$

At inference time, we generate actions by first sampling returns $G_t \sim p_\theta(G_t | s_t, s_G)$ and then sampling actions $a_t \sim \pi_\theta(a_t | s_t, s_G, G_t)$. One can observe that this vanilla sampling procedure recovers the imitative policy since the model will be sampling likely returns under the learned density of the data distribution. Following prior work in offline RL [40, 41, 139], we can also sample actions from an exponentially-tilted policy distribution. This is done by sampling the returns from the tilted distribution

$$G'_t \sim p_\theta(G_t | s_t, s_G) \exp(\kappa G_t) \quad (5.4)$$

with G'_t being the tilted return-to-go and where κ represents the inverse temperature; higher values of κ concentrate more density around the best outcomes or higher returns, while negative values of κ concentrate on less favourable outcomes or lower returns.

For maximal controllability, it is desirable to model the individual components of the reward function rather than their weighted sum. For example, we would like to model an agent's ability to reach its goal, drive on the road, and avoid collisions. Formally, given C reward components, our learned policy should be conditioned on *all* its factored dimensions; this allows us to control each dimension of behaviour separately. This entails modelling separate return components as $G_t^c \sim p_\theta(G_t^c | s_t, s_G)$ for each return component c . Applying this factorization, we can now rewrite the learned policy while explicitly accounting for the conditioning on all return components,

$$\pi_\theta(a_t | s_t, s_G, G_t^1, \dots, G_t^C) \quad (5.5)$$

At test time, each return component will be accompanied by its own inverse temperature κ^c to enable control over each return component, which enables sampling actions that adhere to different behaviours specified by $\{\kappa^1, \dots, \kappa^C\}$, as shown in Algorithm 5 in Appendix B.1.

To implement our framework for multi-agent behaviour simulation, we extend the approach presented above to the multi-agent setting. Across all agents we have sets of the joint states $\mathbb{S}_t$, goal states $\mathbb{S}_G$, actions $\mathbb{A}_t$, and returns-to-go $\mathbb{G}_t$. The final multi-agent joint distribution we model is:

$$p_\theta(\mathbb{S}_{t+1:T}, \mathbb{A}_t, \mathbb{G}_t | \mathbb{S}_t, \mathbb{S}_G) = p_\theta(\mathbb{S}_{t+1:T} | \mathbb{S}_t, \mathbb{S}_G, \mathbb{G}_t, \mathbb{A}_t) \pi_\theta(\mathbb{A}_t | \mathbb{S}_t, \mathbb{S}_G, \mathbb{G}_t) p_\theta(\mathbb{G}_t | \mathbb{S}_t, \mathbb{S}_G), \quad (5.6)$$

where the returns and actions from the previous timesteps are shared across agents, while at the present timestep they are masked out so one can only observe one’s own current return and action.

5.2.2 Multi-Agent Behaviour Simulation Architecture

In this section, we aim to parametrize the multi-agent joint distribution presented in Equation (5.6) for behaviour simulation within the CtRL-Sim framework. To that end, we propose an encoder-decoder Transformer architecture [30], as illustrated in Figure 5.2 where

- The encoder creates a representation of the initial scene, which consists of the initial agents’ states and the map of the environment.
- The decoder autoregressively generates the multi-agent scene rollout, predicting actions and return-to-go values.

Encoder

We are given the initial joint state of all agents in the scene, $\mathbb{S}_0$ as well as their goals $\mathbb{S}_G$, and the map context m . Each agent i ’s initial state s_t^i is composed of its position, velocity, heading and agent type (e.g., vehicle, cyclist, pedestrian), and is encoded using an MLP. The goal state of this agent, s_G^i , is composed of the ground-truth final position, velocity and heading of the agent, and is also encoded using a (different) MLP. We then concatenate these two embeddings and pass them through a linear layer to obtain a final agent embedding of size d . In order to encode agents’ identities throughout the simulation, we employ learnable positional embeddings which are added to the final agent embeddings.

The map context is represented as a set of road segments $m := \{\mathbf{r}_l\}_{l=1}^L$, where each road segment is defined by a sequence of points $\mathbf{r}_l := (p_l^1, \dots, p_l^P)$, where L is the number of road segments and P is the number of points per road segment. We apply a per-point MLP to the points of each road segment $\mathbf{r}_l$. To produce road segment-level embeddings, we then apply attention-based pooling [221] on the embeddings of the points within each road segment, yielding L road segment embeddings of size d . The initial agent embeddings and road segment embeddings are then concatenated into a sequence of length $N + L$ and processed by a sequence of E Transformer encoder blocks.

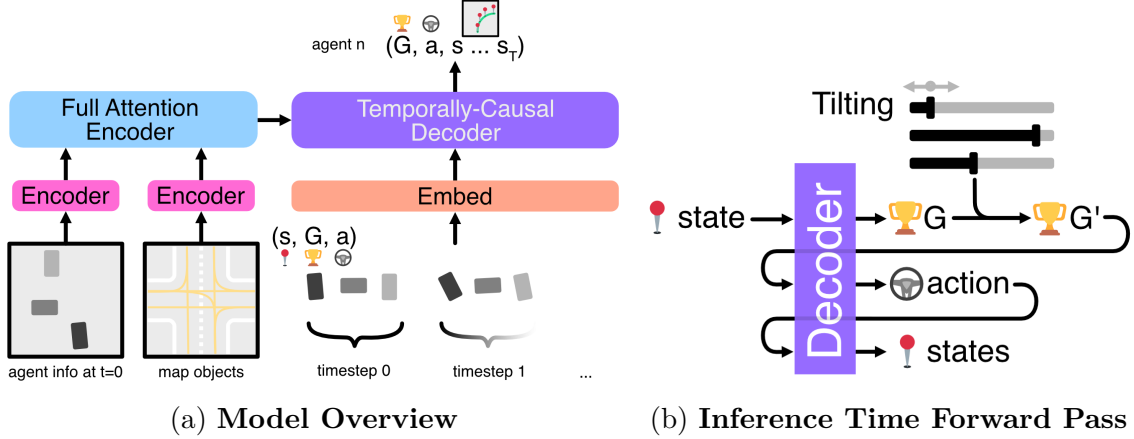


Figure 5.2 **CtRL-Sim Architecture** [26]. 5.2a (left) The agent and map data at $t = 0$ are encoded and fed through a Transformer encoder as context for the decoder, similar to [27]. Trajectories are arranged first by agents, then by timesteps, embedded, and fed through the decoder. For each agent, we encode (s_t, G_t, a_t) (i.e. state, return-to-go, action) and we predict from these $(G_t, a_t, s_{t+1}, \dots, s_T)$. 5.2b (right) At inference time, the state predicts the return-to-go. The return-to-go is tilted (i.e., reweighted to encourage specific behaviors) and is used to predict the action, which in turn is used to predict the next states.

Decoder

We propose a decoder architecture tailored for modelling the joint distribution in Equation (5.6) as a sequence modelling problem. That is, we model the probability of the next token in the sequence conditioned on all previous tokens, $p_\theta(x_t|x_{<t})$ [140]. In this work, our problem can be referred to as a sequential set [18] where our sequences are of the form,

$$x = \langle \dots, (s_t^1, s_G^1), (G_t^{1,1}, \dots, G_t^{C,1}), a_t^1, \dots, (s_t^N, s_G^N), (G_t^{1,N}, \dots, G_t^{C,N}), a_t^N, \dots \rangle \quad (5.7)$$

where s_t^i , s_G^i , and a_t^i correspond to the state, goal, and action of agent i at timestep t , and $G_t^{c,i}$ corresponds to the c 'th factored return component for agent i at timestep t . We note the connection of such sequences to the ones considered in the Multi-Game Decision Transformer [40], but extended to the multi-agent and goal-conditioned case. In contrast to the Decision Transformer [140], CtRL-Sim's architecture predicts the return distribution and samples from it at inference time. We argue that this enables flexible control over the agent behaviours and circumvents the need to specify an expert return-to-go. Moreover, this should ensure that samples remain likely under the data distribution. We obtain state-goal tuple (s_t^i, s_G^i) embeddings in the same way that (s_0, s_G) are processed in the encoder.

Following recent work that tokenizes driving trajectories [27, 222], we discretize the actions

and return-to-go values into uniformly quantized bins. We then embed the action and return-to-go tokens with a linear embedding. To each input token, we additionally add two learnable embeddings representing the agent identity and timestep, respectively. The tokenized sequence is then processed by D Transformer decoder layers with a temporally causal mask that is modified to ensure that the model is permutation equivariant to the agent ordering. We refer the reader to the Appendix B.5 for details.

Training

Given a dataset of offline trajectories (e.g., ones from Section 5.3), we train CtRL-Sim by sampling sub-sequences of length $H \times N \times 3$, where H is the number of timesteps in the context. The state, return-to-go, and action token embeddings output by the decoder are used to predict the next return token, action token, and future state sequence, in that order. This allows us to sample these values while respecting the decomposition of the joint distribution of Equation (5.6). The final loss function is of the form:

$$\mathcal{L} = \mathcal{L}_{\text{action}} + \mathcal{L}_{\text{return-to-go}} + \alpha \mathcal{L}_{\text{state}} \quad (5.8)$$

where $\mathcal{L}_{\text{action}}$ and $\mathcal{L}_{\text{return-to-go}}$ are cross-entropy loss functions, α controls the strength of the auxiliary state prediction loss, and $\mathcal{L}_{\text{state}}$ is an L2 loss function.

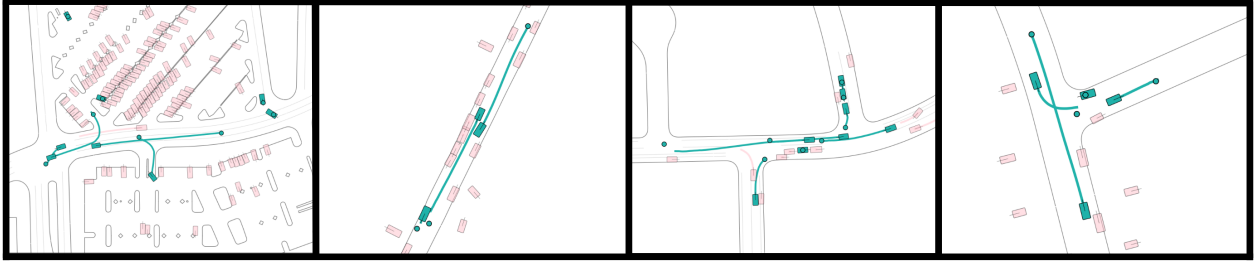


Figure 5.3 **Qualitative results of multi-agent simulation with CtRL-Sim [26].** The teal agents are controlled by CtRL-Sim, and other agents in pink are set to log-replay through physics.

5.3 Experiments

5.3.1 Experimental Setup

In this section, we present the details of the dataset used in our experiments, the evaluation metrics, and the baselines we compare to CtRL-Sim .

Offline RL Dataset

Based on the Waymo Open Motion Dataset (WOMD) [223], we derive a multi-agent offline reinforcement learning dataset. To accomplish this, we extend the Nocturne simulator [38] by incorporating a physics engine based on the Box2D library. As detailed in Appendix B.2.1, this allows us to simulate realistic vehicle dynamics and collisions. Each scene in the WOMD is played back through the physics-enhanced Nocturne simulator, computing the per-timestep actions and factored rewards for each agent. The factored rewards we consider in this work are

- a *goal position* reward, if an agent is within one meter of its goal. The goal is defined as the final ground-truth position.
- a *vehicle-vehicle collision* reward, measured if an agent collides with another agent.
- a *vehicle-road-edge collision* reward, measuring if an agent drives off the road.

The action space is defined by a tuple of the acceleration and steering angle of the vehicle. Similarly to Nocturne [38], we omit scenes containing traffic lights since these are inaccurate in that version of the dataset, and exclude pedestrians and cyclists from the training scenes. As shown in Table 5.1, scenes played back through the physics-enhanced simulator reconstruct adequately the ground-truth scenes, as recorded in the dataset. We provide more details in Appendix B.2.2.

Evaluation

We evaluate CtRL-Sim on (i) *imitation*, its ability to imitate the behaviour policy that generated the Waymo Open Motion driving data, and (ii) *controllability*, its ability to generate counterfactual scenes consistent with specified tilting coefficients. Following the standard Waymo Open Sim Agents Challenge (WOSAC), we adopt the 1-second history given to the model, and simulate the next 8 seconds for all agents. In addition, we run all our evaluations on 1,000 random scenes from the test set. For the *imitation* evaluation, we compute the

relevant metrics for up to 8 agents controlled by CtRL-Sim, where the remaining agents are set to log-replay through physics. We employ the same three main categories of metrics as reported by [130] for evaluating imitation performance:

1. *Reconstruction*: these include average and final displacement errors, and goal success rate.
2. *Distributional Realism*: these use the Jensen-Shannon distances (JSD) between the predicted trajectories and ground-truth.
3. *Common Sense*: these include the rate at which scenarios include instances of collisions or agents veering off the driveable area.

For the controllability experiments, we define one "interesting" interactive agent controlled by the CtRL-Sim agent as an agent who is moving, and whose goal state is within 10 meters of another moving agent. All agents except for the CtRL-Sim-controlled interesting agent are set to log replay through physics. We measure the model’s sensitivity to changing the tilting values of the different reward components and report the corresponding metric. Specifically, we report the goal success rate for the goal reward control, collision rate for the vehicle-vehicle reward control, and offroad rate for the vehicle-road-edge reward control.

Methods under Comparison

For our imitation evaluation, we compare CtRL-Sim to several baselines which have been proposed in the literature. These include:

1. *Replay-Physics* employs an inverse bicycle model to obtain the ground-truth log-replay actions and executes through the simulator. This should recover the ground-truth (GT) trajectory closely as it replays it.
2. *Actions-Only* is an encoder-decoder model inspired by Trajenglish [27] where the decoder trajectory sequences only contain actions. We note that there is a significant difference with Trajenglish as we do not use the k-disks algorithm for actions.
3. *Imitation Learning (IL)* is identical to the architecture in Section 5.2.2 except with the removal of returns and the future state prediction.
4. *Decision Transformer (DT)*: The *GT Initial Return* variant specifies the initial ground-truth return-to-go from the offline RL dataset, with the goal of acting as an imitative policy. This approach is privileged because it uses the GT initial return.

5. *Max Return* follows the standard DT approach of selecting the maximum observable return in the dataset. The DT architecture is identical to that of CtRL-Sim except the return token precedes the state token, and the returns and future states are not predicted by the decoder.
6. *CTG++* is a reimplementation of [132], a competitive Transformer-based diffusion model for behaviour simulation.

For CtRL-Sim ablations, we evaluate several variants:

1. *CtRL-Sim (Base)* as the base CtRL-Sim as described in Section 5.2, trained on the offline RL dataset.
2. A CtRL-Sim model which is trained without the state prediction head, named *CtRL-Sim (No State Prediction)*.
3. *CtRL-Sim (Positive Tilting)* applies $\kappa_c = 10$ tilting to all components c of the CtRL-Sim Base model.
4. *CtRL-Sim (GT Initial Return)* which uses the ground-truth initial return-to-go, and at each timestep decrements the obtained reward from the initial return-to-go until the episode terminates.

In evaluating CtRL-Sim for controllability, we focus on the base model and a finetuned version of the model, (named *CtRL-Sim FT*). Using CAT [224] for collision generation, we finetune a pre-trained base CtRL-Sim model on synthesized long-tail scenarios. We show that this step allows for better controllability as the WOMD’s driving distribution mainly covers nominal driving behaviours. We refer readers to Appendix B.7 for details of CAT and our proposed finetuning procedure.

5.3.2 Results

Imitation

In Table 5.1, we showcase the general imitation results of the baselines and CtRL-Sim. Firstly, we observe that CtRL-Sim models tend to perform competitively against other imitation baseline methods. The CtRL-Sim (Positive Tilting) model achieves the best balance between distributional realism, reconstruction performance and common sense. Its strong performance on the common sense metrics demonstrates the effectiveness of exponential tilting for generating good driving behaviours. Secondly, we note that while imitation learning

Method	ADE (m) ↓	FDE (m) ↓	Goal Success Rate (%) ↑	JSD ($\times 10^{-2}$) ↓	Collision (%) ↓	Off Road (%) ↓	Per Scene Gen. Time (s) ↓
Replay-Physics*	0.47	0.97	87.3	7.6	2.8	10.7	1.1
Actions-Only [27]	4.81 $\pm$ 0.52	11.89 $\pm$ 1.42	32.7 $\pm$ 1.4	10.4 $\pm$ 0.3	19.9 $\pm$ 1.2	27.6 $\pm$ 1.0	3.3
Imitation Learning	1.24 $\pm$ 0.05	1.95 $\pm$ 0.10	77.4 $\pm$ 1.3	8.3 $\pm$ 0.1	5.8 $\pm$ 0.2	12.1 $\pm$ 0.2	3.4
DT (Max Return) [140]	1.56 $\pm$ 0.04	3.07 $\pm$ 0.16	63.3 $\pm$ 0.8	8.4 $\pm$ 0.1	5.3 $\pm$ 0.3	11.0 $\pm$ 0.2	20.7
CTG++ [†] [132]	1.73 $\pm$ 0.10	4.02 $\pm$ 0.32	38.8 $\pm$ 5.4	7.4 $\pm$ 0.2	5.9 $\pm$ 0.4	15.0 $\pm$ 1.5	44.0
CtRL-Sim (No State Prediction)	1.32 $\pm$ 0.03	2.21 $\pm$ 0.06	72.4 $\pm$ 0.8	8.2 $\pm$ 0.2	6.1 $\pm$ 0.4	12.0 $\pm$ 0.3	8.2
CtRL-Sim (Base)	1.29 $\pm$ 0.04	2.13 $\pm$ 0.08	73.0 $\pm$ 1.3	8.1 $\pm$ 0.2	5.8 $\pm$ 0.4	11.8 $\pm$ 0.2	
CtRL-Sim (Positive Tilting)	1.25 $\pm$ 0.03	2.04 $\pm$ 0.08	72.9 $\pm$ 1.5	7.9 $\pm$ 0.1	5.3 $\pm$ 0.2	11.0 $\pm$ 0.2	
DT* (GT Initial Return)	1.10 $\pm$ 0.02	1.58 $\pm$ 0.07	77.5 $\pm$ 1.5	8.4 $\pm$ 0.1	5.3 $\pm$ 0.3	11.9 $\pm$ 0.3	20.8
CtRL-Sim* (GT Initial Return)	1.09 $\pm$ 0.02	1.60 $\pm$ 0.06	77.2 $\pm$ 1.1	8.1 $\pm$ 0.2	5.6 $\pm$ 0.4	12.2 $\pm$ 0.1	

Table 5.1 **Multi-agent simulation results over 1000 test scenes [26]**. We report mean $\pm$ std across 5 seeds. CtRL-Sim achieves a good balance between reconstruction performance, common sense, realism, and efficiency. * indicates privileged models requiring GT future. [†] indicates reimplementations.

(IL) performs similarly to CtRL-Sim and has better inference time (since it decodes fewer tokens), a major disadvantage is its lack of controllability. While DT (Max Return) exhibits strong performance on the common sense metrics, it performs poorly on the reconstruction metrics compared to CtRL-Sim models. Finally, we also confirm that the prediction of the future state rollout, i.e., CtRL-Sim (Base), performs (slightly) better than CtRL-Sim (No State Prediction) baseline across all metrics.

Controllability

One of the two main objectives of CtRL-Sim is intuitive controllability over agents’ behaviours. This is enabled through exponential tilting of the return distributions of the individual components. While DT can in theory generate suboptimal behaviours by specifying low initial return-to-go values, we argue that a user would have difficulty in calibrating this value as it demands prior knowledge of the problem domain and lacks an intuitive knob for smooth behaviour changes. As discussed in Section 5.2, CtRL-Sim provides a clear interpretation and smooth knob over behaviours in the form of exponential tilting:

- A negative tilt will push the behaviour distribution towards worse than average behaviours observed in the data.
- Conversely, a positive tilt will yield behaviours better than the average.

We show the results of our controllability evaluation in Figure 5.4. For each reward dimension c , we exponentially tilt κ_c between -25 and 25 and observe how this affects the corresponding

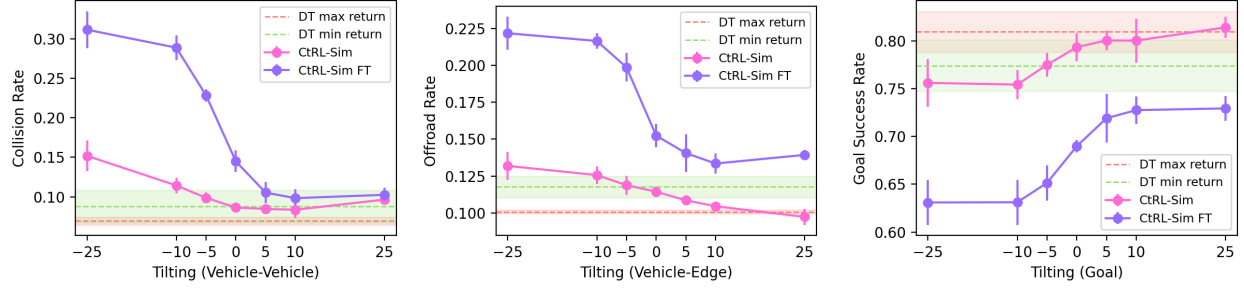


Figure 5.4 **Effects of exponential tilting** [26]. Comparison of CtRL-Sim base model (magenta) and fine-tuned model (purple) across different reward dimensions. Rewards range from -25 to 25 for vehicle-vehicle collision (**left**), vehicle-edge collision (**middle**), and goal reaching (**right**). Results show smooth controllability, with fine-tuning enhancing this effect. We report mean $\pm$ std over 5 seeds.

Adv. Method	Tilt	Reactive?	Control?	Planner Metrics		Adversary Realism	
				Progress (m) $\downarrow$	Coll. w/ Adv. (%) $\uparrow$	JSD ($\times 10^{-2}$) $\downarrow$	Coll. Speed (m/s) $\downarrow$
CAT		$\times$	$\times$	53.3	61.4	18.7	6.9
CtRL-Sim	-10	$\checkmark$	$\checkmark$	57.5 $\pm$ 0.1	10.0 $\pm$ 0.5	10.8 $\pm$ 0.3	7.4 $\pm$ 0.5
	10			57.7 $\pm$ 0.1	8.7 $\pm$ 0.5	10.3 $\pm$ 0.5	8.3 $\pm$ 0.6
CtRL-Sim FT	-10			56.1 $\pm$ 0.2	33.8 $\pm$ 1.9	17.4 $\pm$ 0.6	6.3 $\pm$ 0.2
	10	$\checkmark$	$\checkmark$	57.1 $\pm$ 0.1	18.5 $\pm$ 1.6	12.6 $\pm$ 0.7	6.1 $\pm$ 0.2
	50			57.4 $\pm$ 0.2	12.8 $\pm$ 0.2	15.6 $\pm$ 1.2	6.0 $\pm$ 0.3

Table 5.2 **Adversarial scenario generation results over 1000 test scenes** [26]. We report the mean $\pm$ std over 5 seeds for the CtRL-Sim models. Finetuning CtRL-Sim on CAT data improves the ability to generate adversarial scenarios compared with the base CtRL-Sim model. Compared with CAT, CtRL-Sim is reactive and controllable, while exhibiting better collision realism.

metric of interest. We also show the results of DT when conditioning on the minimum and maximum possible return.

As we hypothesized, we observe a monotonic change in each of the relevant metrics corresponding to the return-to-go component as the tilting value increases. We note the importance of the fine-tuning procedure of CtRL-Sim on long-tail scenarios as it enhances the overall policy’s ability to generate negative long-tail scenarios. Specifically, at -25 tilting, the finetuned model is able to generate $2.1\times$ as many collisions and $1.8\times$ as many offroad violations as the base model. Figure 5.5 (and B.1, B.2 in Appendix B.8) show qualitatively the effects of tilting.

Effects on a Planner

In this section, we focus on CtRL-Sim’s ability to produce adversarial agents that interact, and possibly collide, with a data-driven planner. Table 5.2 shows the results on a held-out test set of two-agent interactive scenarios from the Waymo interactive dataset. In this setting, one agent is controlled by the adversary policy while the other is controlled by the data-driven planner. We note that the data-driven planner in our experiments is a positively tilted CtRL-Sim base model, since it exhibited good driving behaviour in Table 5.1. For the adversarial policies, we evaluate both the base and the finetuned CtRL-Sim models. With -10 tilting applied, the finetuned model generates 238 more collisions with the planner than the base model over 1000 scenes, which we attribute to its exposure to simulated collision scenarios during finetuning. We note that finetuning was performed in only 30 minutes on a single NVIDIA A100-large GPU, with only 3500 CAT scenarios. This showcases the potential for scaling CtRL-Sim on new data sources through finetuning, which would enable generation of a wide-range of driving behaviours. While CtRL-Sim does lose some of its ability to generate good driving behaviours compared to the base model, we can see that, with a +50 tilting value, the model can reduce its collision rate with the planner by 21.1%.

We also compare CtRL-Sim against a state-of-the-art collision generation method, CAT [224], which uses a motion prediction model to select plausible adversarial trajectories that overlap with the ego plan. Although CAT generates more collisions, it has two major shortcomings. Firstly, CAT is *not controllable*; it cannot control how adversarial the agents are. Secondly, and crucially, CAT agents are *non-reactive* to the ego’s actions; their trajectories are generated and fixed prior to the simulation, which greatly hinders their ability for wide adoption. This is evidenced by a larger adversary collision speed than all finetuned CtRL-Sim models.

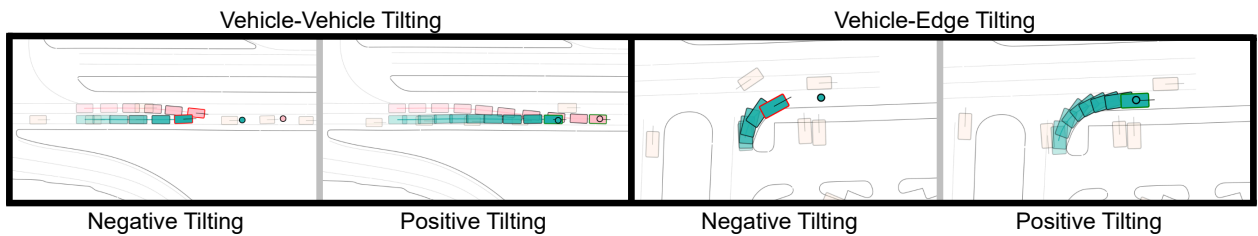


Figure 5.5 **Qualitative results of vehicle-vehicle and vehicle-edge tilting [26]**. Two traffic scenes comparing positive vs. negative tilting of a CtRL-Sim-controlled agent (shown in teal). Bounding boxes in red indicate traffic violations. Other agents are log-replay through physics, with interacting agents in pink. Goals are marked by small circles.

5.4 Conclusion

In this chapter, we presented CtRL-Sim, a novel framework applying offline RL for controllable and reactive behaviour simulation. Our proposed multi-agent behaviour Transformer architecture allows CtRL-Sim to employ exponential tilting at test time to simulate a wide range of agent behaviours. We presented experiments showing the effectiveness of CtRL-Sim at producing controllable and reactive behaviours, while maintaining competitive performance on the imitation task compared to baselines.

Limitations The learned policies produced by CtRL-Sim in its current form may make large-scale RL training on CtRL-Sim scenarios prohibitively expensive. However, we believe that further optimizations of the model and simulator such as exploring more compact models, utilizing model quantization or Flash Attention [225], reducing the planning frequency, or making the simulator GPU-accelerated [28] can further improve the inference latency. In addition, an interesting extension to CtRL-Sim is to condition the policy on language descriptions of what happens in the scenario [132]. We leave these extensions for future work. In addition, CtRL-Sim is a data-driven approach that relies on the map information and initial agent configuration. We believe that for large-scale data-driven simulation, we will need models that can synthesize the entire scenario, including the initial scene. We propose an approach in the next chapter.

Finally, Table 5.1 highlighted that the usage of privileged GT initial return information provides a strong prior for simulated rollouts. We argue that this observation points to a key limitation of CtRL-Sim: the factorization of the return components assumes an independence that is not necessarily true. Specifically, CtRL-Sim treats the return-to-go distributions as follows:

$$p_{\theta}(G_t^{\text{coll}}, G_t^{\text{goal}}, G_t^{\text{offRoad}} | s_t, s_G) = \prod_{k \in \{\text{coll}, \text{goal}, \text{offRoad}\}} p_{\theta}(G_t^k | s_t, s_G), \quad (5.9)$$

However, we note that these components are coupled by shared dynamics and absorbing events which violate this independence assumption. For example, aggressive maneuvers can increase the likelihood of reaching the goal, but this would co-vary with higher collision probability and/or off-road risk. In general, there exists a small set of latent factors (e.g., scene difficulty, road geometry, traffic density) that would jointly modulate these rewards in a more faithful manner; we leave the exploration of such factors to future work.

CHAPTER 6 SYNTHESIZING INITIAL SCENES FOR FULLY DATA-DRIVEN SIMULATION

In the previous chapter, we presented our CtRL-Sim framework for modelling reactive and controllable behaviour simulation. One of its limitations is its reliance on the pre-recorded initial scene configurations, limiting our ability to scale training and evaluation to new environments. Prior work for synthesizing initial scene configurations employed rasterization of the vectorized map, and modelled the data distribution with the parameter-heavy image-based diffusion models. In this chapter, we introduce a novel vectorized latent diffusion model for initial scene generation that directly operates on the vectorized scene elements. We demonstrate that this vectorized scene-generation approach attains superior generation quality while using fewer parameters, offering substantially lower generation latency, and requiring markedly fewer GPU-hours for training than the strongest image-based approaches. Moreover, we show that the model’s training can be extended to allow it to learn how to inpaint; this allows us to extrapolate to generating large scenes. Combining this proposed initial scene generation and CtRL-Sim from Chapter 5, we propose Scenario Dreamer, a fully data-driven closed-loop generative simulator that supports training and evaluation of existing AV planners. We validate its practical utility by showing that pretrained PPO agents struggle more in *Scenario Dreamer*-generated environments than in traditional non-generative simulators, with the gap widening on long routes and adversarial scenarios. The contributions of this chapter are:

1. We introduce a novel vectorized latent diffusion model for initial scene generation. It offers practical computational advantages as well as produces better quality over prior methods that utilize a rasterized scene encoding.
2. Combining the vectorized latent diffusion model with CtRL-Sim from Chapter 5, we present Scenario Dreamer, a fully data-driven *generative* simulator for autonomous driving planning.
3. We show that Scenario Dreamer environments challenge RL planners, especially on long and adversarial driving environments.

The contents of this chapter make reference to content from the following publication:

L. Rowe, R. Girgis, A. Gosselin, L. Paull, C. Pal, and F. Heide, “Scenario dreamer: Vectorized latent diffusion for generating driving simulation environments,” CVPR, pp. 17 207–17 218, 2025 (© 2025 IEEE)

6.1 Introduction

Simulators are essential for the safe and scalable deployment of autonomous vehicles (AVs) [72, 73, 226, 227]. They play a key role in exposing AV planners and policies to rare and hazardous conditions, and therefore can lower the overall testing costs and ensure robustness. Although simulators have become much faster and more realistic recently, they remain largely reliant on pre-recorded driving logs [43, 86, 87], which mostly consist of nominal driving behaviours. As we saw in Chapter 5, offline reinforcement learning can be an effective way of extracting long-tail behaviours grounded in real-world logs. In this chapter, we argue that the reliance on driving logs is fundamentally limiting for a few reasons.

Firstly, the average person drives each year roughly 22,000 km [228] while the Waymo Open Motion dataset (WOMD;[223]), which is used in GPU Drive[87], Waymax[43] and Nocturne[38] simulators, only covers 1,750 km of unique roads. As noted by Cusumano et al. [229], the WOMD maps are small and are composed of incomplete road networks constructed from several US cities. In order to support scalable driving policies, it is imperative that we expose planners and driving policies to a wider range of road geometry, which we cannot necessarily obtain through recorded logs unless one has access to a costly network of data collectors. While OpenStreetMap [230] provides broad geographic coverage, its maps are often too coarse for high-fidelity simulation, lacking reliable lane-level geometry, traffic-control detail, and dynamic context; moreover, relying on existing maps or logs still requires searching for rare structures, such as roundabouts, rather than directly generating targeted road configurations. While one could in theory send data collectors to record and annotate logs from said desirable road structures, we argue that generative models provide a more efficient solution.

The second problem is the lack of adversarial agents within them. While there has been a recent surge of methods for long-tail behaviour simulation [26, 128, 224], one is always limited by the data distribution. For example, in our CtRL-Sim work [26], while one can simulate adversarial behaviours through exponential tilting, we find that this approach can easily lead to degenerative behaviours where agents seek to collide rather than collisions happening “organically”. When a road accident happens, it is likely the case that it was due to an error of judgment rather than the colliding agent seeking to cause a crash. Many existing approaches tackle the problem of long-tail scenario generation with realism in mind [68, 128].

Finally, safety validation of autonomous vehicle planners in simulation requires simulation of billions of miles of diverse driving experiences[231, 232]. It is important to know how an

agent would interact with other agents it shares the road with for extended periods. Current approaches to data-driven behaviour simulators are bounded by the length of the driving logs, which are usually less than 30 seconds. This inhibits progress in supporting scalable training and evaluation of autonomous vehicles. Ideally, agents are placed in infinitely large simulated worlds where they are allowed to drive for long stretches, providing autonomous driving companies with valuable data on how the autonomy stack behaves in the simulated-wild.

In this Chapter, we present a first step towards a simulation framework that solves all three of the problems listed above. Our solution, called Scenario Dreamer [44], is a framework for synthesizing new road structures in which we can simulate agents. Recent work has shown the successful usage of diffusion generative models for synthesizing driving environments comprised of bird’s eye view (BEV) agent bounding boxes and lane graph [92, 93]. However, these methods encode the driving scenes as rasterized BEV images, which makes them unnecessarily costly as compute is wasted on empty pixels in the scene [92, 93]. Furthermore, these approaches do not directly address the problem of fully data-driven agent behaviours within the simulation environments: [93] employs unrealistic rule-based models, while [92] only generates short, non-reactive agents rollouts.

Scenario Dreamer [44] is a fully data-driven *generative* simulator to train and evaluate autonomous driving planners. We decompose the data-driven simulator into two separate conceptual components. The first focuses on the initial scene generation problem including the state of all agents in the scene and the road structure within a field-of-view (FOV). For this, we propose a vectorized latent diffusion model that operates on vectorized scene elements directly. We show that this approach is more efficient than rasterization for the problem of initial scene generation and offers higher quality scene extrapolation in dense road geometries. Furthermore, thanks to the vectorized representation, we can explicitly control scene density by specifying the number of lanes and agents within a given field of view (FOV). Learning to generate the map in a vectorized manner allows us to learn and use the lane graph connectivity, eliminating the need for post-processing heuristics used by *sledge* [93].

The second component of Scenario Dreamer tackles the problem of behaviour simulation where we build on CtRL-Sim [26] from Chapter 5. Equipped with our behaviour simulation approach, we inherit the advantages associated with exponential tilting [26, 40] which enable test-time control for generating safety-critical scenarios. Both models are trained separately on real-world driving data. We show that Scenario Dreamer provides more challenging environments for RL planning agents, trained in GPUDrive [87], compared to existing non-generative environments. This work unlocks the ability to synthesize a limitless quantity of

unbounded and interactive safety-critical environments, which we hope stimulate future autonomous driving planning research. Through its inpainting, we show that Scenario Dreamer can also generate large maps, extending beyond the original FOV.

6.2 Proposed Framework

We approach generative driving simulation by decomposing it into *initial scene generation* and *behaviour simulation*. This section details our initial scene generator trained on real-world data, followed by a generative simulation framework for closed-loop evaluation of AV planning agents. Initial scene generation produces BEV object bounding boxes and the map within a fixed FOV. Following [93], we use a $64\text{m} \times 64\text{m}$ FOV centered and yaw-aligned to the ego. Let F denote this region, and let F_P and F_N be the $32\text{m} \times 64\text{m}$ areas ahead of and behind the ego, respectively. A scene generator samples initial scenes $\mathcal{I}_F = \{\mathcal{O}, \mathcal{M}\}$ from $p(\mathcal{I}_F)$. Here $\mathcal{O} = \{\mathbf{o}_i\}_{i=1}^{N_o}$ contains traffic participants (*e.g.*, ego, vehicles, pedestrians, cyclists) and static objects (*e.g.*, cones); each $\mathbf{o}_i \in \mathbb{R}^8$ stores 2D position, speed, cos / sin heading, length, width, and class. We represent the map as $\mathcal{M} = \{\mathcal{L}, \mathbf{A}\}$, following [93], where $\mathcal{L} = \{\mathbf{l}_i\}_{i=1}^{N_l}$ are N_l centerlines with $\mathbf{l}_i \in \mathbb{R}^{20 \times 2}$, and $\mathbf{A} \in \{0, 1\}^{N_l \times N_l \times 4}$ encodes successor, predecessor, left, and right adjacency. To support arbitrarily long simulations, the generator also samples $p(\mathcal{I}_{F_P} \mid \mathcal{I}_{F_N})$, enabling stitching of new F_P to existing F_N (see Figure 6.3).

6.2.1 Vectorized Latent Diffusion for Initial Scene Generation

Our goal is to model data-generating distribution of the initial scene, $p(\mathcal{I}_F)$. We propose learning an approximation from real-world driving scenes using a diffusion model, $\tilde{p}(\mathcal{I}_F) \approx p(\mathcal{I}_F)$. Diffusion models have shown strong capability in capturing highly complex distributions such as generating images or video [233, 234]. In this chapter, we approach this problem using a two-stage training process: (1) we learn a compact latent representation of the N_o objects, $\{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o}$, and the N_l centerlines, $\{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l}$, through a low- β variational autoencoder [235]; (2) we train a diffusion model to sample from the latent distribution $p(H) := p(\{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o}, \{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l})$. We refer the reader to Chapter 2 for our presentation on both components. Figure 6.1 presents an overview of our architecture.

Autoencoder

The autoencoder is based on Transformer blocks [236] for the encoder $\mathcal{E}_\phi$ and decoder $\mathcal{D}_\gamma$. These operate directly on vectorized scene elements, unlike [93] which rasterizes the scene. The encoder embeds the $N_o + N_l$ vectors with per-vector MLPs and embeds one-hot lane-

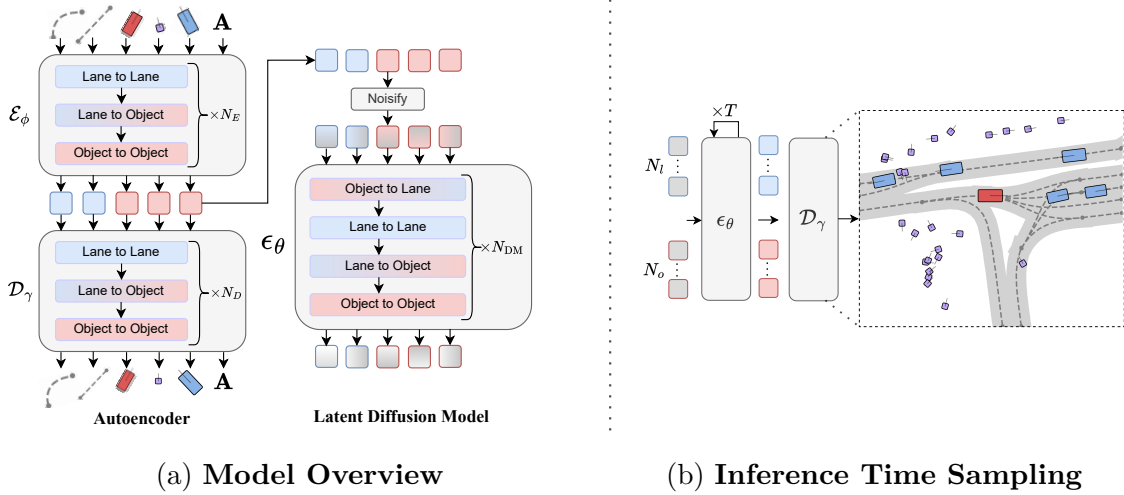


Figure 6.1 **Overview of initial scene generation with our vectorized latent diffusion model** [44]. *Left*: Each vectorized scene element is embedded into a latent representation with a variational autoencoder parameterized with factorized attention blocks. The lane attention additionally fuses the lane connectivity information, $\mathbf{A}$. The latent Transformer diffusion model ϵ_θ is trained to sample from the autoencoder’s latent distribution. *Right*: Scenario Dreamer samples novel driving scenes by randomly initializing $N_o + N_l$ noise tokens which are iteratively denoised with ϵ_θ over T steps and decoded into vectorized scene elements. The ego vehicle is denoted in red, with other agents colored in blue and pedestrians in purple. (© 2025 IEEE)

connectivity types c_{ij} for all centerline pairs (i, j) . It then applies N_E factorized attention blocks for lane-to-lane, lane-to-object, and object-to-object relationships with dedicated multi-head attention layers [109, 110, 237], as in Chapter 4. The lane-to-lane attention layers are responsible for capturing spatial relationships between different centerline segments, and they fuse the connectivity embeddings into keys/values. The lane-to-object attention layers inject map context into the object embeddings, and object-to-object attention layers capture inter-object relationships. The output of the encoder, $\mathcal{E}_\phi$, maps lanes and objects latent embeddings into compressed latent dimensions K_l and K_o with mean/variance parameterization as in a VAE [238] from Chapter 2. An important feature of the encoder is its avoidance of object-to-lane attention layers, effectively ensuring that lane latents are independent of object features; this enables lane-conditioned object generation at inference. The decoder $\mathcal{D}_\gamma$ samples latents $\{\{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o}, \{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l}\} \sim \mathcal{E}_\phi$, processes them through N_D factorized attention blocks, and reconstructs continuous lane/object vectors with an ℓ_2 loss. The lane connectivity is predicted by concatenating the embeddings of lane-pairs and applying an MLP with cross-entropy supervision. Training uses the standard ELBO with low- β regularization.

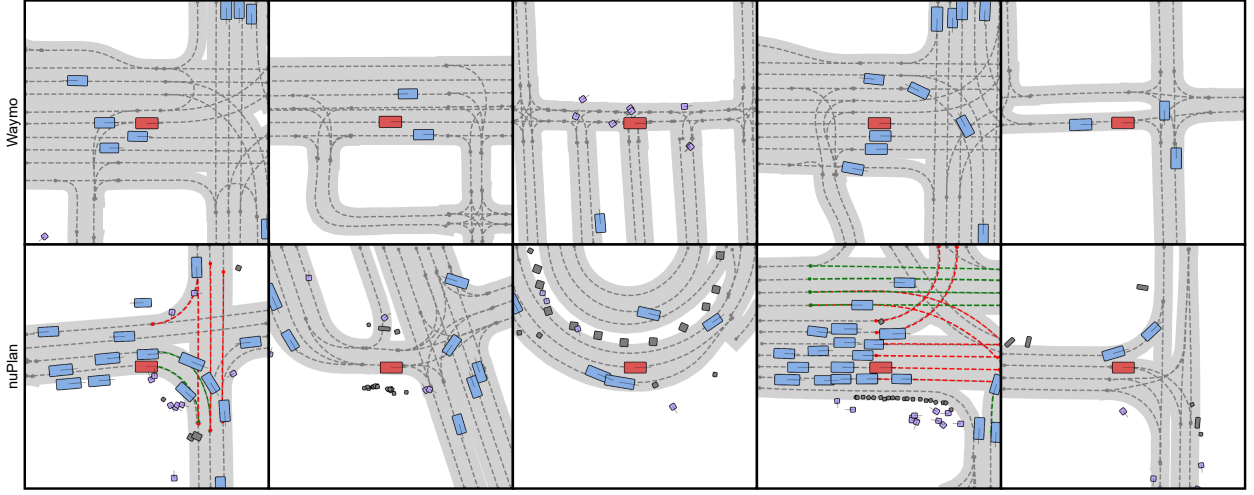


Figure 6.2 **Qualitative results of Vectorized Environment Generation [44]**. Vectorized environments generated by Scenario Dreamer with the proposed vectorized latent diffusion model trained on the Waymo dataset (top row) and nuPlan dataset (bottom row). (© 2025 IEEE)

Latent Diffusion Model

We train a latent diffusion model to sample from the latent distribution induced by the autoencoder, $p(H)$, which we factorize as:

$$p(H) = \sum_{N_o, N_l} p(\{\mathbf{h}_i^O\}_{i=1}^{N_o}, \{\mathbf{h}_i^L\}_{i=1}^{N_l} | N_o, N_l) p(N_o, N_l). \quad (6.1)$$

$$\approx \sum_{N_o, N_l} p_\theta(\{\{\mathbf{h}_i^O\}_{i=1}^{N_o}, \{\mathbf{h}_i^L\}_{i=1}^{N_l}\} | N_o, N_l) p(N_o, N_l) \quad (6.2)$$

where $p(N_o, N_l)$ is approximated with training set statistics, and we approximate $p(H)$ as $p(H) \approx p_\theta(H)$. We model the conditional distribution $p_\theta(\cdot | N_o, N_l)$ with a diffusion model with weights θ , which samples N_o object latents and N_l lane latents for the autoencoder’s decode to use. Unlike image-based models, we deal with variable set sizes; p_θ is designed with a customized transformer architecture which employs AdaLN-Zero conditioning [239]. Similarly to the general theme of this thesis, this architecture is composed of a sequence of N_{DM} *factorized* attention blocks (illustrated in Figure 6.1), which are composed of object-to-lane, lane-to-lane, lane-to-object, and object-to-object attention layers. This factorized processing approach allows each layer to model the layer-specific interactions while allowing for different hidden dimensions for object and lane tokens for an efficient design. That is, since the road geometry requires a high level of spatial reasoning and detail compared to the object tokens, we set a larger hidden dimension for the lane latents, while object tokens are represented with a smaller dimension. We employ the standard DDPM objective to train p_θ , where p_θ is parameterized as a noise-prediction network ϵ_θ that learns to predict the noise of

noised lane and object latents at varying noise levels:

$$L_{\text{dm}} = \mathbb{E}_{\mathbf{H}_t, \boldsymbol{\epsilon}_t \sim \mathcal{N}(0,1), t} \left[\|\boldsymbol{\epsilon}_t - \epsilon_\theta(\mathbf{H}_t, t)\|_2^2 \right],$$

where $\mathbf{H}_t$ denotes a stacked tensor of N_o object latents and N_l lane latents that is noised over t forward diffusion steps with noise vector $\boldsymbol{\epsilon}_t := (\boldsymbol{\epsilon}_{\mathcal{L}}, \boldsymbol{\epsilon}_{\mathcal{O}})$ containing both the object and lane noise vectors. We refer readers to the Appendix for more details.

Addressing Permutation Ambiguity

One issue with Vectorized diffusion is the phenomenon called *permutation ambiguity* [106]: when vector elements are excessively noised during training, e.g., with large t values, they lose the underlying structure such that the model cannot correctly determine the permutation of the ground-truth signal. In image-based transformer diffusion approaches (e.g., DiT), positional encodings are applied to different grid patches. In a similar vein, we address permutation ambiguity by introducing sinusoidal positional encodings to latent tokens ahead of the factorized attention blocks. Our positional encoding scheme involves defining an *ordering* of the tokens (per token type) during training. This allows the model to better infer a noised token’s relative spatial position and thus the ground-truth signal toward which it should regress. To achieve this, we propose a recursive ordering procedure: Specifically, our positional encoding follows a recursive procedure: tokens are ordered by minimum x -value, and if x -values differ by less than ϵ meters, they are subsequently ordered by minimum y -value, then maximum x -value, and finally maximum y -value. This design choice has the useful byproduct of facilitating inpainting: tokens in $\mathcal{I}_{F_N}$ form a contiguous subsequence of ordered tokens that precede those in $\mathcal{I}_{F_P}$, recalling that F_N and F_P are the regions behind and ahead of the ego-agent.

Scene Generation Modes

Our proposed vectorized latent diffusion model supports multiple modes of scene generation within a single trained model:

1. **Initial scene generation.** To sample $64\text{m} \times 64\text{m}$ scenes from $p(\mathcal{I}_F)$, our vectorized latent diffusion model first samples $(N_o, N_l) \sim p(N_o, N_l)$ from the joint distribution found in the training data, with dataset-dependent limits on N_o and N_l to ensure realistic scene density. A user can also exhibit a bit of control over the scenes being generated by manually specifying (N_o, N_l) . After setting (N_o, N_l) , latents $\{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o}, \{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l}$ are

sampled using the diffusion model over $T = 100$ reverse diffusion steps, followed by a single inference call to the Autoencoder’s decoder, $\mathcal{D}_\gamma$, for the final vectorized scene elements.

2. **Lane-conditioned object generation.** To sample object bounding boxes $\mathcal{O}$ conditioned on a known map $\mathcal{M}$, we first encode the vectorized elements of a road map $\mathcal{M}$ using the autoencoder’s encoder, $\mathcal{E}_\phi$. We then sample $N_o \sim p(N_o|N_l)$ object latents from the diffusion model; these are diffused while being conditioned on the encoded map latents at each denoising timestep. The generated object latents can be decoded into object bounding box configurations with the autoencoder’s decoder, $\mathcal{D}_\gamma$.
3. **Scene inpainting** In order to sample the area in front of the ego conditioned on the area behind the ego, $p(\mathcal{I}_{F_P}|\mathcal{I}_{F_N})$, we first note that unlike the grid-structured image domain with boundaries naturally defined (e.g., along $x = 0$; Sledge [93]), vectorized lanes can easily be on both sides of such a boundary. We preprocess scenes into *partitioned* (artificially split at $x=0$) and *non-partitioned* (lanes may cross $x=0$). Our autoencoder is trained to reconstruct both scene types, while the diffusion model is trained with a conditioning label to distinguish them. We found that inpainting quality significantly improved when we condition on the encoded latents of $\mathcal{I}_{F_N}$ during training, as this explicitly teaches the model to inpaint and yields lane geometries that remain consistent across the $x=0$ boundary. Our diffusion model requires specifying the number of new lanes and objects to occupy the new 32mx64m region, $\mathcal{I}_{F_P}$. We train a classifier, f_ϕ , to predict the number of lanes in front of the ego, $N_l^{F_P}$, conditioned on the map behind the ego-agent, $\mathcal{M}_{F_N}$. At inference, we first sample $N_l^{F_P} \sim f_\phi$, followed by sampling $N_o^{F_P} \sim p(N_o^{F_P}|N_o^{F_N}, N_l^{F_N} + N_l^{F_P})$. We then generate $\mathcal{I}_{F_P}$ as expected: (1) encode the latents for $\mathcal{I}_{F_N}$ using the encoder; (2) initialize $N_o^{F_P}$ and $N_l^{F_P}$ tokens to Gaussian noise; and (3) generate the vectorized scene $\mathcal{I}_{F_P}$ through T diffusion steps, ensuring that tokens in $\mathcal{I}_{F_N}$ are set to their un-noised encoded latents at each denoising step. An example of an inpainted scene is shown in Figure 6.3.

6.2.2 Full Scenario Simulation Framework

Recall from Chapter 5 our behaviour simulation framework, CtRL-Sim [26], which employs return-conditioned supervised learning to control multiple agents conditioned on an initial scene $\mathcal{I}$, formulated as

$$p_\theta(\mathbb{A}_t, \mathbb{G}_t|\mathbb{S}_t, \mathcal{I}) = \pi_\theta(\mathbb{A}_t|\mathbb{S}_t, \mathbb{G}_t, \mathcal{I})p_\theta(\mathbb{G}_t|\mathbb{S}_t, \mathcal{I}). \quad (6.3)$$

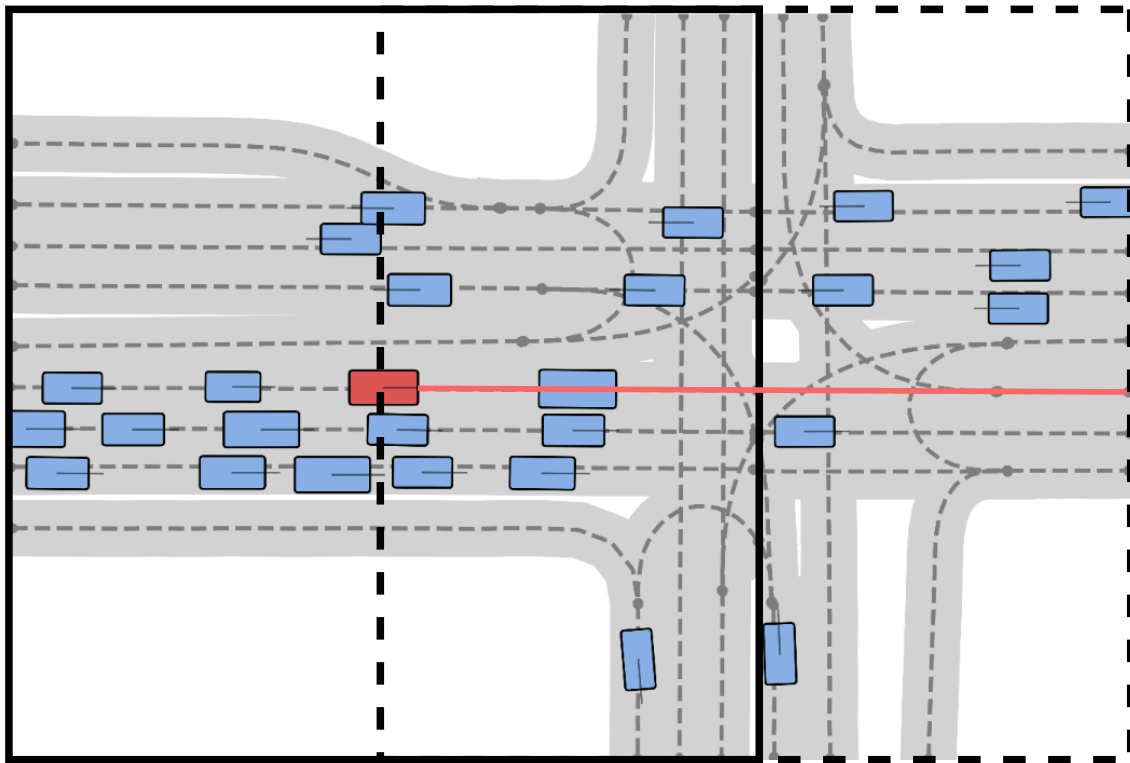


Figure 6.3 **Example of Scenario Dreamer Inpainting** [44]. Illustrative example of Scenario Dreamer’s inpainting capabilities, where the initial tile is outlined in solid lines and the inpainted tile in dashed. The model generates consistent lane geometries at the scene boundaries, even at complex intersections. (© 2025 IEEE)

We remind the reader that a key advantage of CtRL-Sim’s decomposition is it enables exponential tilting [40] of the learned return distribution at inference to generate good or adversarial driving behaviours.

In this chapter, we couple our vectorized latent diffusion model with the CtRL-Sim model to form our full scenario simulation framework, which we call *Scenario Dreamer*. We note that we employ a slightly modified version of CtRL-Sim in this Chapter, removing the reliance on the Nocturne environment [38] for speed of simulation. In addition, we adopt the use of a k -disks tokenization scheme introduced by Pillion et al. [240]. The goal of our Scenario Dreamer framework is to create adversarial scenarios that specifically challenge the ego agent (e.g., an AV planner). To this end, our CtRL-Sim model is trained with a reward function that penalizes collisions with the ego vehicle.

Scenario Dreamer supports the evaluation of AV planners within its generative simulation environments over arbitrarily long simulation lengths thanks to the CtRL-Sim behaviour simulation and the vectorized latent diffusion model’s inpainting ability. In such environments, we define a route for the AV planner to follow, and simulate the other agents using CtRL-Sim. We follow the Waymo dataset filtering scheme employed in Nocturne [38] and GPUDrive [87] to ensure that the scenarios are valid for simulation (*i.e.*, absence of traffic lights, which lack annotated traffic light states). We provide more details on the simulation framework in the Appendix.

6.3 Experiments

6.3.1 Experimental Setup

In this section, we present the details of the dataset used in our experiments, the evaluation metrics we employ, and the baselines we compare to the Scenario Dreamer Framework.

Datasets

We train the vectorized latent diffusion model separately on both the Waymo Open Motion Dataset [223] and the nuPlan dataset [86]. Our model is trained on vehicles, pedestrians, and cyclists, and we solely focus on the lane centerlines and their connectivity, excluding other map elements for both datasets. We note that there is no barrier in theory to adding the road edges to the architecture, but opted against it as our focus in this work is multi-agent interaction; this is in line with SLEDGE’s approach [93]. In addition, for nuPlan, we also have access to the traffic light states and therefore include these as vectorized element types.

For both datasets, we use a $64\text{m} \times 64\text{m}$ FOV centered on the ego vehicle and centerlines are processed using the compression algorithm from [93]; details are provided in the Appendix.

Metrics

We assess vectorized latent diffusion on both lane graph generation and initial agent bounding box generation tasks. For lane graph generation, we report **Urban Planning** metrics, following prior works [92, 93, 102]. Such metrics focus on distributional realism of lane graph connectivity using the Frechet distances on node features from nodes with degree $\neq 2$; these are referred to as *key points*. To this end, the following features are computed: *Connectivity* computes the degrees of all key points across the lane graphs; *Density* computes the number of key points in each lane graph; *Reach* computes the number of available paths from each keypoint to others; and *Convenience* computes the Dijkstra path lengths for all valid paths between key points. As in SLEDGE [93], the Urban Planning metrics are scaled by suitable powers of 10 for readability. **Frechet Distance (FD)** measures the perceptual quality of the generated spatial lane positions by computing Frechet distances between the penultimate layer lane embeddings of a separately trained Scenario Dreamer autoencoder model. We additionally measure the longest **Route Length** by traversing the generated lane graph from the origin, and **Endpoint Distance**, which measures the average distance between the endpoint of lane i and starting point of lane j for all predicted successor edges (i, j) in the generated lane graphs. All lane graph generation metrics are computed on 50k generated lane graphs and 50k ground-truth test lane graphs.

For initial agent generation, we focus on the distributional realism of agent placements, reporting Jensen-Shannon Divergence (**JSD**) metrics as in [101]. These include: *Nearest Distance* between each vehicle and its neighbours, *Lateral Deviation* from the closest centerlines, *Angular Deviation* from the closest centerline, *Length* and *Width* of the vehicles, and *Speed* of the vehicles. The JSD metrics are computed over 50k real and generated scenes and scaled by suitable powers of 10. We also report the **Collision Rate** of the agent bounding boxes in the initial scene. Behaviour simulation metrics are similar to those of the agent generation, and to the ones from Chapter 5. We present the specific details in the Appendix.

Methods under Comparison

On *nuPlan*, we compare against SLEDGE (both DiT-L and DiT-XL) [93], retraining their models using the official open-source code. SLEDGE rasterizes driving scenes to an image latent, models that latent with a latent diffusion model, and then decodes back to a vectorized representation. On the Waymo dataset, we use DriveSceneGen [92] as the baseline. We note

that the DriveSceneGen diffusion model did not converge; therefore, to provide an *upper bound* on its performance, we rasterized 50k ground-truth training scenes with their pipeline and applied DriveSceneGen’s vectorization post-processing. Our proposed vectorized latent diffusion is the only method that processes scene elements fully in vector space end-to-end.

For behaviour simulation, we compare CtRL-Sim against the rule-based IDM [32] and a data-driven baseline Trajenglish [240], with quantitative and qualitative results reported in the Appendix. The Scenario Dreamer latent diffusion model is available in two sizes: the base model (B) with 377M parameters, trained over 24 hours on 4 A100 GPUs, and the large model (L) with 679M parameters, trained over 32 hours on 8 A100 GPUs. Additional model specific details are provided in the Appendix.

6.3.2 Results

Initial Scene Generation

Table 6.1 summarizes lane-graph results on nuPlan and Waymo, and reports per-scene generation time, parameter counts, and diffusion training GPU-hours (GPUh). On nuPLAN, *Scenario Dreamer* outperforms SLEDGE across all metrics. Interestingly, *Scenario Dreamer (Base)* surpasses SLEDGE’s largest DiT-XL on every metric while using $\sim 2\times$ fewer parameters, $\sim 7\times$ lower inference latency, and $\sim 10\times$ fewer GPUh. Our reimplement of SLEDGE DiT-XL underperforms the scores reported by [93]; nevertheless, *Scenario Dreamer (L)* exceeds the published SLEDGE results on all metrics.

Dataset	Method	Per-Scene	Num.	Perceptual Quality ↓		Urban Planning ↓				Route Length ↑	Endpoint Dist. ↓
		Gen. Time (s) ↓	Parameters	GPUh	FD	Conn.	Dens.	Reach	Conve.	(m)	(m)
nuPlan	SLEDGE (DiT-L) [93]	0.48	539M	96	1.89	2.34	2.43	0.70	2.44	35.34±8.63	0.47±0.32
	SLEDGE (DiT-XL) [93]	0.67	769M	960	1.44	1.67	1.74	0.51	1.68	35.83±8.35	0.42±0.29
	Scenario Dreamer (B)	0.09	377M	96	1.05	0.28	0.45	0.07	0.14	37.04 ±10.21	0.32±0.80
	Scenario Dreamer (L)	0.16	679M	256	0.67	0.18	0.43	0.03	0.33	36.87±10.37	0.25 ±0.71
Waymo	DriveSceneGen (GT Raster)* [92]	-	-	-	40.59	4.53	1.18	0.64	5.58	41.61 ±18.61	0.01 ±0.00
	Scenario Dreamer (B)	0.08	376M	96	1.61	0.17	1.05	0.56	3.81	38.23±12.78	0.32±0.90
	Scenario Dreamer (L)	0.16	678M	256	1.38	0.03	0.95	0.28	2.05	38.92±13.56	0.21±0.75

Table 6.1 **Quantitative Assessment of Lane Graphs** [44]. Assessment of lane graph generation evaluated on the Waymo Open Motion and nuPlan test datasets. For each metric and dataset, the best method is **bolded**. * denotes a privileged version of the method. (© 2025 IEEE)

The improved inference time and superior lane generation quality highlight the effectiveness of the vectorized architectural prior. Vectorized processing of the road structure focuses the parameter capacity on relevant scene elements since each latent corresponds to a specific scene element, avoiding wasteful capacity modelling empty pixels. Practically, this results in

Scenario Dreamer’s latent diffusion model requiring far fewer Transformer layers to model effectively compared to SLEDGE. Concretely, the largest SLEDGE diffusion model requires 28 Transformer layers for processing, whereas Scenario Dreamer’s largest latent diffusion model requires only 6 lane-to-lane attention layers.

Table 6.2 presents the results on agent generation on the nuPlan and Waymo datasets. As indicated by the lower JSD metrics and fewer collisions compared to baselines, Scenario Dreamer is better at generating realistic initial agents. In the Appendix, we visualize a random sample of Scenario Dreamer, SLEDGE, and privileged DriveSceneGen generated scenes, where Scenario Dreamer scenes exhibit visually more realistic lane graph and agent configurations.

Dataset	Method	Distributional JSD ↓						Collision Rate ↓
		Near. Dist.	Lat. Dev.	Ang. Dev.	Len.	Wid.	Speed	(%)
nuPlan	SLEDGE (DiT-L) [93]	0.53	0.63	3.60	11.84	10.16	0.46	22.3
	SLEDGE (DiT-XL) [93]	0.49	0.49	3.26	11.16	10.29	0.47	21.2
	Scenario Dreamer (B)	0.12	0.15	0.17	0.22	0.14	0.07	11.9
	Scenario Dreamer (L)	0.09	0.11	0.18	0.25	0.17	0.06	9.3
Waymo	DriveSceneGen (GT Raster)* [92]	0.63	1.01	2.43	58.86	54.51	18.70	0.2
	Scenario Dreamer (B)	0.07	0.07	0.07	0.40	0.25	0.36	5.4
	Scenario Dreamer (L)	0.06	0.05	0.07	0.42	0.25	0.38	4.8

Table 6.2 **Quantitative Assessment of Agent Placement** [44]. Assessment of initial agent bounding box generation evaluated on the Waymo Open Motion and nuPlan test datasets. * denotes privileged version of the method. (© 2025 IEEE)

Table 6.3 summarizes ablations on key architectural choices. Replacing the learned lane-connectivity head with a heuristic based on endpoint distance and relative orientation (*Heuristic topology*) degrades the Reach and Connectivity Urban Planning metrics, indicating that it is better to learn the lane connectivity directly rather than using post-processing heuristics. Removing our lane-ordering scheme (*No lane ordering*) introduces permutation ambiguity during training and inference, reducing generation quality as hypothesized above. Finally, a non-factorized variant with a comparable parameter count (*Non-factorized*) both underperforms and increases inference latency by roughly $2\times$, underscoring the accuracy and efficiency benefits of the factorized design.

Method	Per-Scene Gen. Time (s)	Perceptual Quality ↓	Urban Planning ↓				Route Length ↑	Endpoint Dist. ↓
		FD	Conn.	Dens.	Reach	Conve.	(m)	(m)
Scenario Dreamer (Non-factorized)	0.20	1.21	0.51	0.67	0.16	0.55	36.71±9.95	0.37±0.76
Scenario Dreamer (No lane ordering)	0.09	1.36	0.60	0.07	0.21	0.71	36.70±9.94	0.35±0.74
Scenario Dreamer (Heuristic topology)	0.09	1.05	0.28	0.44	0.12	0.60	37.13±9.85	0.30±0.23
Scenario Dreamer	0.09	1.05	0.28	0.45	0.07	0.14	37.04±10.21	0.32±0.80

Table 6.3 **Ablation of Vectorized Latent Diffusion** [44]. Ablation experiments validating the design choices of Vectorized Latent Diffusion (B) for lane graph generation on the nuPlan dataset. (© 2025 IEEE)

Impact on RL Planner

Table 6.4 summarizes the performance of a PPO [62] planner (see Chapter 2 for details) trained in GPU Drive [87] on 100 Waymo training scenes, reformatted to be compatible with Scenario Dreamer simulation environments. The policy is trained to follow a set of waypoints, defined as a route while avoiding collisions. We evaluate the planner in (i) generative Scenario Dreamer environments with 55 m (*SD (55m)*) and 100 m (*SD (100m)*) routes, and (ii) non-generative Waymo test scenes (*Waymo Test*). For consistency, Waymo maps are reduced to centerlines only. We report collision rate (*Coll.*), off-road rate (*Offroad*; lateral deviation > 2.5 m from the route), and success rate (*Succ.*; route completion). Replacing non-reactive log-replay agents with reactive CtRL-Sim behaviors yields only marginal changes in the RL policy’s performance (Row 1 vs. Row 2), underscoring the realism of CtRL-Sim. Evaluating on *SD (55m)*, which matches the average Waymo route length, produces comparable planner performance (Row 1 vs. Row 2), further affirming Scenario Dreamer’s fidelity. Increasing the route length to 100 m notably degrades performance (Row 2 vs. Row 3), and negative tilting in CtRL-Sim exacerbates this effect (Row 3 vs. Row 4), indicating that longer and more adversarial scenarios are more challenging for the planner.

Other Agent Behaviour	Test Environment	Avg. Route Length (m)	Collision (%)	Offroad (%)	Success Rate (%)
Log Replay	Waymo Test	55	29.3 $\pm$ 0.6	6.9 $\pm$ 0.8	63.8 $\pm$ 0.8
CtRL-Sim (Pos. Tilt)	Scenario Dreamer (55m)	55	33.8 $\pm$ 1.2	6.4 $\pm$ 0.7	59.8 $\pm$ 1.7
CtRL-Sim (Pos. Tilt)	Scenario Dreamer (100m)	100	52.8 $\pm$ 1.4	9.1 $\pm$ 1.6	38.2 $\pm$ 1.9
CtRL-Sim (Neg. Tilt)	Scenario Dreamer (100m)	100	59.0 $\pm$ 1.3	9.0 $\pm$ 0.8	32.1 $\pm$ 1.3

Table 6.4 **RL Planner Results** [44]. PPO agents are trained in GPU Drive on 100 Waymo scenes and evaluated on 250 scenes (Waymo test scenes or 55m/100m-route Scenario Dreamer (SD) scenes). mean $\pm$ std reported over 5 seeds. (© 2025 IEEE)

6.4 Conclusion

In this chapter, we present Scenario Dreamer, a fully data-driven generative simulator enabling the evaluation of autonomous driving planners. The key novelty of Scenario Dreamer is our proposed vectorized latent diffusion model that efficiently models the initial scene as a set of vectorized elements. Coupling the diffusion model with CtRL-Sim from Chapter 5 allows us to simulate long-tail environments within large maps through inpainting. We hope our approach to simulation can serve as a foundation for future work in data-driven approaches.

Limitations and Future Work A first limitation of our work is in the problem of large-scale map generation. While we partially address this problem in Scenario Dreamer with inpainting, we believe this is still limiting since CtRL-Sim agents cannot persist beyond the

edge of the map. Furthermore, it is not trivial to perform loop closure of the map generation because of the *behind-ahead-of-ego* construction. We believe this is an important avenue for future work in order to faithfully support long-horizon simulation for training or evaluation. Another important limitation is our restriction to only generating centerlines ignoring other road structures, e.g., road boundaries. While this should not be a difficult to integrate with vectorized latent diffusion, one must be careful with certain road boundary polylines as they do not always adhere to a *start-end* structure, e.g., roundabouts. In addition, in our nuPlan experiments, we observed that traffic lights signalization of lanes did not provide valid traffic logic. Scenario Dreamer also does not account for varying weather conditions, which can affect road-surface friction and agent behaviours; incorporating such conditions would be an interesting direction for future work. Finally, an interesting extension would be to ground the conditioning variables for initial scene generation in coarse road topology from OpenStreetMap [230], while using the generative model to synthesize simulation-ready lane geometry and targeted road structures.

CHAPTER 7 SPECIFYING DESIRABLE BEHAVIOURS WITH CONSTRAINED REINFORCEMENT LEARNING

Reinforcement learning for autonomous driving has been explored as a potentially powerful framework for generating good driving. However, the standard Reinforcement Learning formulation of maximizing a single scalar reward does not afford designers the ability to specify behavioural constraints. The most common approach is to manually tune the weights of a reward function, a process that could demand many iterations with expensive experimentation. Furthermore, this process may give rise to agents that “hack” the reward function—i.e., exploit misspecified proxies or metric loopholes to maximize nominal reward via behaviours misaligned with the designer’s intent. In this chapter, we advocate for the use of constrained reinforcement learning as a practical way to substantially reduce the effort required to specify the behaviours we want agents to exhibit. Specifically, we use Lagrangian methods to automatically weigh each of the behavioural constraints agents should respect. We showcase our approach in the realm of NPC design in video games with SAC as the basis of our algorithm. We extend our method to the realm of end-to-end autonomous driving by finetuning large pre-trained vision-language models with constrained-GRPO, showcasing the general utility of this framework. The contributions of this chapter are:

1. We demonstrate that reward engineering does not scale gracefully with increasing task complexity.
2. We introduce a formulation that allows designers to directly specify the desired frequency of key events, offering a more interpretable and intuitive way of guiding agent behaviour than reward shaping alone.
3. We develop a novel reinforcement learning algorithm capable of jointly satisfying a large number of constraints without collapsing performance on the main RL task.
4. We expand our proposed framework to constrain RL-finetuned large pre-trained vision-language models (VLMs) with GRPO, named *Constrained GRPO*. We study the common practice of scalarizing rewards prior to normalization, and prove that it implicitly reweights objective components based on within-group variances and inter-term correlations, thereby distorting the relative importance intended by the Lagrangian multipliers.
5. For constrained GRPO, we empirically demonstrate improved stability and faithfulness

of the behaviour specification in a gridworld environment and a realistic simulation-based autonomous driving setting with a VLM-based planner. We further validate this framework with a constrained version of soft Actor-Critic [59] on a set of constrained tasks illustrative of the development cycle required for deploying RL agents in video games in Appendix D.

7.1 Introduction

Reinforcement Learning (RL) has shown rapid progress and led to many successful applications over the past years [65, 241, 242, 243]. The central idea of RL is that the success of an agent on any specific task can be measured by a single scalar function, and its task is to maximize that function; this is generally referred to as the reward hypothesis [58, 244, 245]. While the idea of a single reward function has proven quite useful in developing the theoretical framework, it can be significantly limiting to force all desirable information from real-life problems into one single reward function. In practice, human-designed reward functions often lead to unforeseen behaviours and represent a serious obstacle to the reliable application of RL in the industry [246].

Concretely, for a designer working on applying an RL method to a real-world problem, the task of reward specification implies one can :

1. Describe mathematically the desired behaviour that the agent should maximize.
2. Write a computer program for which the optimal policy generates the desired behaviour.
3. Train an RL agent on that task using one of the methods available in the literature.
4. Evaluate whether the trained agent exhibits the expected behavioural characteristics.

In practice, it is difficult to specify that reward function to produce the desired behaviour, which leads to many costly design iterations and policy training [161, 247]. Further exacerbating the issue is the fact that most Deep RL algorithms cannot be guaranteed to find the optimal policy [58]. In other words, even if the reward function is correctly specified, it can still lead to undesirable behaviours. An example of the type of behaviors a designer might be seeking is shown in Figure 7.1. We argue that the same challenge arises in driving: rather than relying on delicate reward tuning, we should encode key safety and behavioral requirements as constraints (Figure 7.2).

Most of the literature in reinforcement learning focuses on the problem of improving the reliability and efficiency of algorithms that yield a (near-)optimal policy given the reward

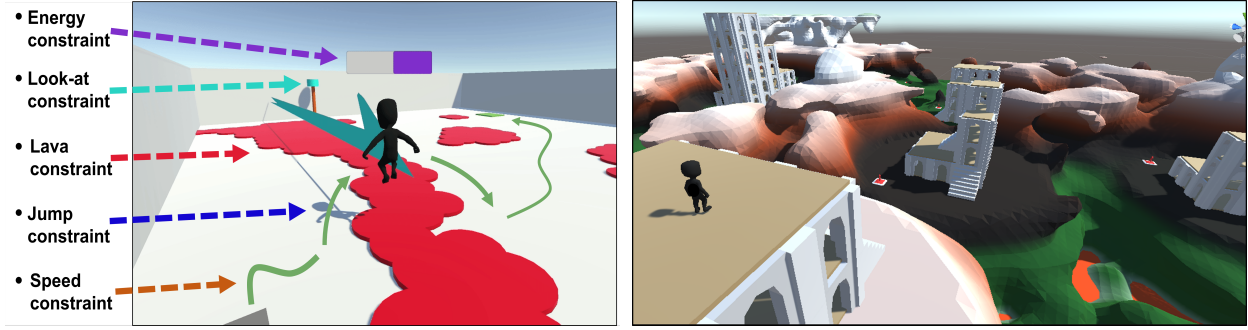


Figure 7.1 **Overview of Environments** [25]. Depictions of our setup to evaluate direct behaviour specification using constrained RL; Arena environment (left); OpenWorld environment (right). We can see that agents are faced with large environments that they must navigate while respecting certain behavioural constraints. In this chapter, we proposed the use of constrained RL in order to learn policy that adheres to behaviour specifications. For more information see: <https://sites.google.com/view/behaviourspecificationviacrl/home>.

function. Of course, this is an important and difficult problem that demands attention from researchers. However, as argued by Daniel Dewey [24], modern AI research will need to contend with the problem of reward design in order to obtain policies that “do the right thing”.

Constrained Markov Decision Processes (CMDPs) [145] offer an alternative framework for sequential decision making. In CMDPs, the agent still seeks to maximize a scalar reward function, but the feasible policy space is restricted to those that satisfy a set of constraints, each expressed through cost functions. We note that it is generally recognized that this framework offers an easier way for the end user to define their tasks [155]. However, most work focuses on the issue of constraint-satisfying behaviours maintaining “safe” exploration [149, 152, 248, 249]. In this chapter, we specifically present the benefits of CMDPs in regard to direct behaviour specification [25], and how it extends to constraining reinforcement finetuning of large multimodal models with constrained GRPO [250].

7.2 Empirical Analysis of Reward Engineering

We start with an empirical study that showcases the impracticality of using reward engineering to generate desired behaviours, motivating the need for the direct behaviour specification framework. We consider a simple navigation task in which the agent is tasked with reaching a goal location while being subject to additional behavioural constraints. These constraints are (1) looking at a visible marker 90% of the time, (2) avoiding forbidden terrain 99% of the time and (3) avoiding running out of energy also 99% of the time. The environment



Figure 7.2 **Driving Environment Overview** Example scene of the driving problem and the context the agent is presented with. The agent does not perceive privileged information, i.e., the bounding boxes in the images and the vectorized scene representation in the center are only provided for illustrative purposes. In our experiments, the agent is limited to only observing the three front views, due to computational limitations, along with its speed and intention as textual prompts.

is depicted in Figure 7.1 (left) and the details are presented in Appendix D.2. The reward function for this task is of the form:

$$R'(s, a) = R(s, a) - \mathbf{1} * w_{\text{not-looking}} - \mathbf{1} * w_{\text{in-lava}} - \mathbf{1} * w_{\text{no-energy}} \quad (7.1)$$

where $R(s, a)$ gives a small shaped reward for progressing towards the goal and a terminal reward for reaching the goal, and the $\mathbf{1}$'s are indicator functions which are only active if their corresponding behaviour is exhibited.

A central difficulty in reinforcement learning practice lies in **tuning the weights of auxiliary reward components**, e.g., $w_{\text{not-looking}}$, $w_{\text{in-lava}}$, $w_{\text{no-energy}}$, so that the agent achieves its primary objective (reaching the goal) while simultaneously respecting additional behavioural requirements. This tuning process is commonly known as *reward engineering*. If these weights are set too low, the agent disregards the constraints altogether; if set too high, the agent becomes overly constrained and fails to solve the main task. Striking the right balance is rarely intuitive, and in practice is often approached through repeated trial-and-error across the space of coefficients $\{w_k\}$.

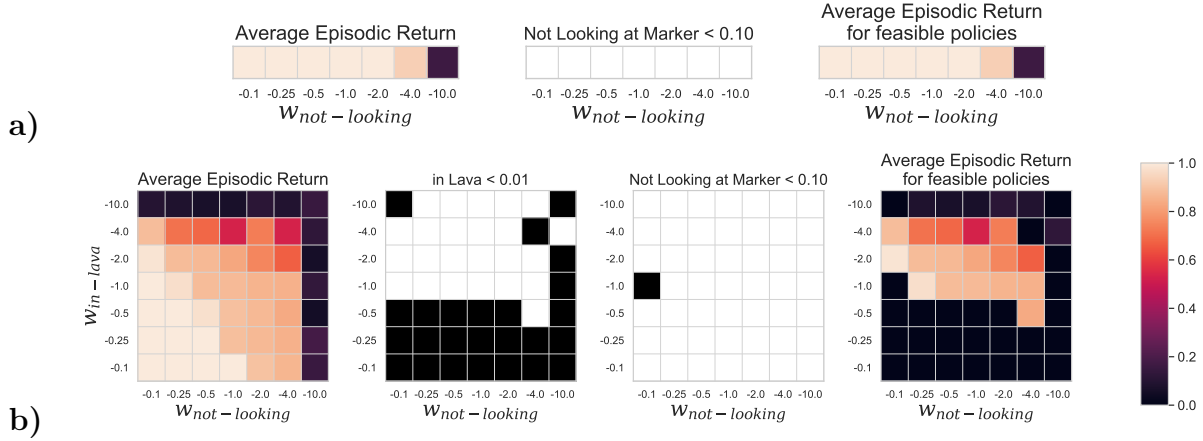


Figure 7.3 Exposing reward engineering with one and two constraints [25]. Enforcing behavioural constraints using reward engineering. Each grid represents a different metric. Within each grid, each square represents the final performance (according to that metric) of an agent trained for 3M steps using the reward function in Equation 7.1 parameterised as given by the grid coordinates. Performance is obtained by evaluating the agent on 1000 episodes. The leftmost column indicates the episodic return of the trained policies, the middle columns indicate whether or not the agent respects the behavioural constraint(s) and the rightmost column indicates the average return for these feasible policies only. **a)** The “looking-at marker” behaviour does not affect too much the main task and, consequently, all chosen weights allow the agent to satisfy the constraint (looking at marker 90% of the time) and many of them also lead to good performance on the main navigation task ($-0.1 \geq w \geq -2$). **b)** When also enforcing the “Not in Lava” behaviour, which is much more in the way of the main task, most of the resulting policies do not respect the constraint or perform poorly on the navigation task, highlighting the difficulty of choosing the correct penalty weights ahead of time. Out of 49 experiments, only two yielded good performing feasible policies: $(-0.10, -2.0)$ and $(-0.25, -1.0)$. In the largest search with 3 behavioural constraints, none of the 343 experiments found a good performing feasible policy (see Figure 7.4).

To highlight the difficulty, we perform grid searches over three settings: a single constraint, two constraints, and three constraints. For each case, we sweep 7 values per weight, ranging from 0.1 to 10 times the scale of the main reward. This results in 7, 49, and 343 separate training runs, respectively. The outcomes (Figures 7.3 and 7.4) show that as the number of constraints increases, the fraction of runs yielding a successful policy decreases dramatically. One can imagine that a grid-search becomes more and more infeasible as the size of the policy increases, as in our VLM for driving experiments in Section 7.5.2. In practice, the engineer is forced into an undesirable trade-off: rely on ad-hoc manual tuning guided by intuition, or resort to an increasingly expensive grid search. While expert heuristics or more advanced search strategies can mitigate the burden to some extent, the approach of reward engineering does not scale with task complexity.

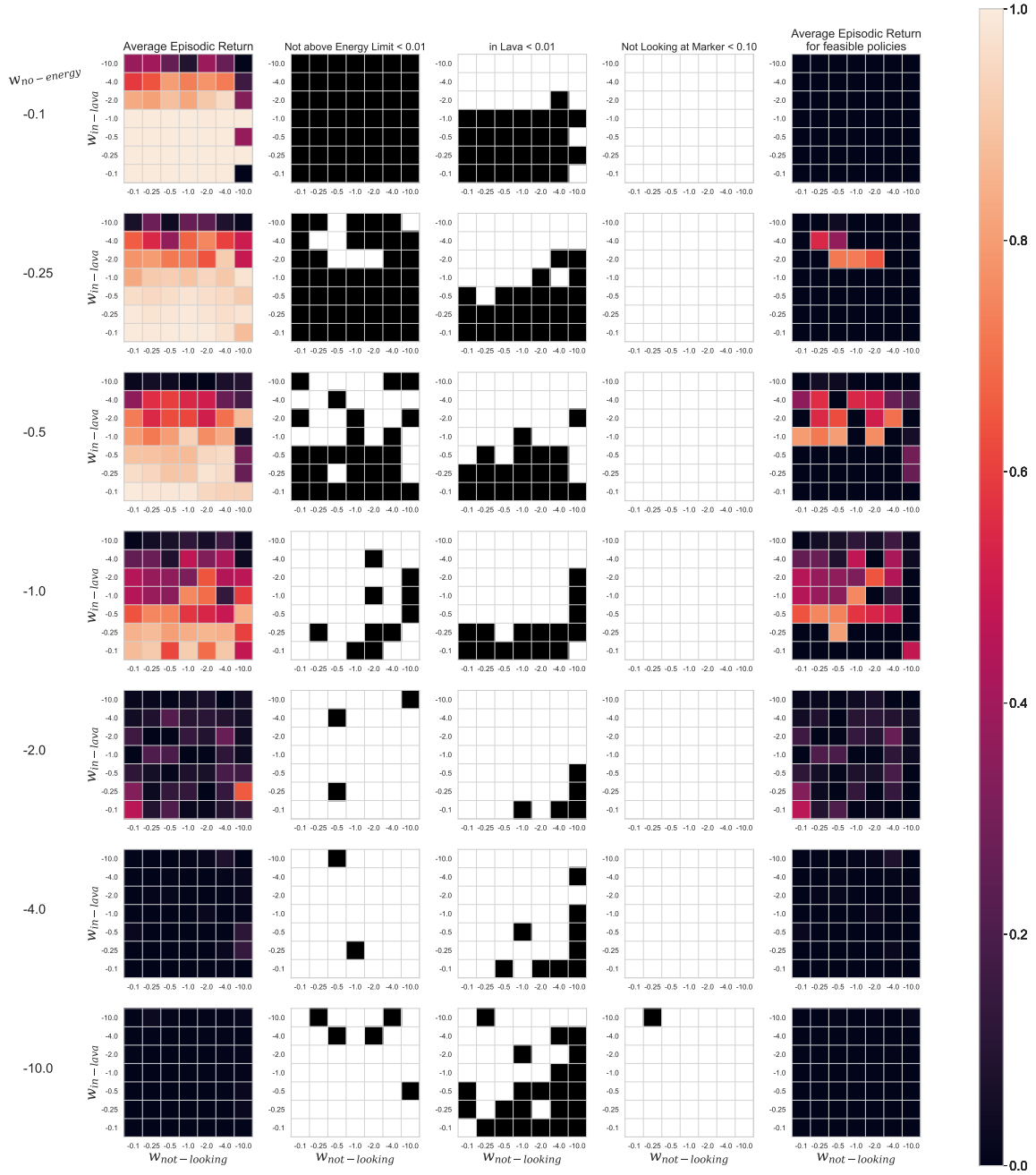


Figure 7.4 **Exposing reward engineering with three constraints [25]**. An even larger proportion of the experiments are wasted finding either low-performing policies or policies that do not satisfy the behavioral constraints. In this case, none of the 343 experiments yielded a feasible policy that also solves the task (success rate near 1.0), showcasing that reward engineering scales poorly with the number of constraints due to the curse of dimensionality and to the compounding effect of the multiple constraints in narrowing the space of feasible policies.[25]

It is worth emphasizing that even if the *reward hypothesis* holds in principle, i.e., that all tasks can be cast as maximizing a scalar reward function [58], this does not imply that the function itself admits a simple or human-intuitive form. Complex, multi-faceted behaviours may only be expressible through reward functions whose structure exceeds the limits of human intuition [245]. In the following sections, we introduce a practical framework based on constrained Markov decision processes (CMDPs), which offers a more interpretable and human-centric interface for specifying behavioural requirements.

7.3 Constrained Markov Decision Processes

Constrained Markov Decision Processes (CMDPs) [145] extend the standard MDP framework to settings where multiple cost constraints must be satisfied in addition to the main objective. Formally, we define a set of cost functions $C_k : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, each associated with a threshold $d_k \in \mathbb{R}$ that must not be exceeded in expectation. The set of feasible policies is defined as

$$\Pi_C = \{\pi \in \Pi : J_{C_k}(\pi) \leq d_k, \quad k = 1, \dots, K\}, \quad (7.2)$$

and the optimal policy is the one that maximizes the expected return within this feasible set:

$$\pi^* = \operatorname{argmax}_{\pi \in \Pi} J_R(\pi) \quad \text{s.t.} \quad J_{C_k}(\pi) \leq d_k, \quad k = 1, \dots, K. \quad (7.3)$$

Unlike standard MDPs, where it is sufficient to restrict the search to stationary deterministic policies, optimal policies in CMDPs may require considering the broader space of stationary *stochastic* policies to guarantee optimality [145].

A widely used family of algorithms for solving CMDPs relies on *Lagrangian relaxation*, which has been shown to produce good feasible solutions in practice [148, 149, 152, 155]. This approach is based on the saddle point characterization of optimal solutions for nonlinear programs with inequality constraints [251, 252, 253]. The central idea is to combine the main RL objective and constraints into a single Lagrangian function:

$$\max_{\pi} \min_{\lambda \geq 0} \mathcal{L}(\pi, \lambda), \quad \mathcal{L}(\pi, \lambda) = J_R(\pi) - \sum_{k=1}^K \lambda_k (J_{C_k}(\pi) - d_k), \quad (7.4)$$

where $\lambda = \{\lambda_k\}_{k=1}^K$ are nonnegative Lagrange multipliers that determine the relative importance of each constraint. Uzawa's method [251] solves this min-max problem by alternating between gradient ascent on the policy parameters and gradient descent on the multipliers. This gradient ascent-descent procedure [254] is conceptually similar to the optimization dy-

namics of Generative Adversarial Networks [255].

The maximization of the Lagrangian with respect to the policy can be interpreted as optimizing an unconstrained objective with a modified reward function

$$L(s, a) = R(s, a) - \sum_{k=1}^K \lambda_k C_k(s, a), \quad (7.5)$$

allowing any standard policy optimization method to be applied directly. For the multipliers, the gradient simplifies to

$$\nabla_{\lambda_k} \mathcal{L}(\pi, \lambda) = -(J_{C_k}(\pi) - d_k), \quad (7.6)$$

and a projected gradient step ensures $\lambda_k \geq 0$ via a max-clipping operator. Intuitively, if a constraint is violated ($J_{C_k}(\pi) > d_k$), the corresponding multiplier λ_k increases, which raises the penalty in $L(s, a)$ and encourages the policy to reduce constraint violations. Conversely, if a constraint is satisfied ($J_{C_k}(\pi) < d_k$), the multiplier decreases, allowing the optimization process to shift focus toward other constraints or the main reward function R .

7.4 Direct Behaviour Specification

In most reinforcement learning benchmarking environments, researchers can focus on improving current algorithms at finding better policies, faster and more reliably, with the assumption that the reward function is provided and its maximization results in a desirable policy. In industrial settings, multiple behavioral desiderata are typically imposed on the agent, and combining these objectives into a single reward function is often challenging and non-trivial. In the following sections, we present a framework that leverages CMDPs to specify such behaviors more efficiently.

7.4.1 Indicator cost functions

The difficulty of specifying the desired behaviour of an agent using a single reward function arises primarily from the need to carefully tune the relative weights assigned to each reward component. As the number of components grows, identifying appropriate relative scales becomes increasingly complex (see Section 7.2). While the prioritization and saturation properties of CMDPs can help decompose the behavioural specification problem [155], significant design challenges remain.

First, CMDPs allow for arbitrary cost function definitions, which can unintentionally lead to undesirable or unforeseen behaviours. Second, determining appropriate constraint thresh-

olds d_k is often non-trivial and difficult to do based solely on intuition. For example, in their Mujoco experiments, Zhang et al. [152] ran an unconstrained version of PPO [62] to estimate typical cost magnitudes before applying their constrained solver with manually selected thresholds.

In this chapter, we propose a simple way to bypass this separate threshold-estimation phase by focusing on a subclass of CMDPs that enables a more intuitive connection between the cost functions C_k and their expected returns J_{C_k} . Specifically, we restrict cost functions to be *indicator functions* of the form:

$$C_k(s, a) = I(\text{behaviour } k \text{ is met in } (s, a)), \quad (7.7)$$

which simply expresses whether the agent exhibits behaviour k when taking action a in state s . An appealing property of this design is that the expected discounted sum of indicator costs can be re-expressed as an expectation over the visitation distribution, allowing us to interpret $J_{C_k}(\pi)$ as a *re-scaled probability* of exhibiting behaviour k during interaction with the environment:

$$J_{C_k}(\pi) = \mathbb{E}_{\tau \sim p_\pi} \left[\sum_{t=0}^T \gamma^t C_k(s_t, a_t) \right] \quad (7.8)$$

$$= Z(\gamma, T) \mathbb{E}_{(s,a) \sim x_\pi(s,a)} [C_k(s, a)] \quad (7.9)$$

$$= Z(\gamma, T) \Pr_{(s,a) \sim x_\pi} (\text{behaviour } k \text{ is met in } (s, a)). \quad (7.10)$$

By dividing both sides of the constraint $J_{C_k}(\pi) \leq d_k$ by $Z(\gamma, T)$, we obtain a normalized threshold $\tilde{d}_k \in [0, 1]$ that represents the *desired frequency* of behaviour k under the policy.

In practice, we compute the empirical mean of the indicator costs over the batch, giving equal weight to all state-action pairs regardless of their temporal position:

$$\tilde{J}_{C_k}(\pi) := \frac{1}{N} \sum_{i=1}^N C_k(s_i, a_i), \quad (7.11)$$

where i indexes samples in the batch. For numerical stability, we train the corresponding critic $Q^{(k)}$ with a discount factor $\gamma_k < 1$, when a critic is used.

Although indicator cost functions are simple as defined in Equation 7.7, they are expressive enough to capture a wide range of behavioural preferences. Crucially, they provide *clear semantics* for setting constraint thresholds as probabilities and enable intuitive, low-effort behaviour specification by simply choosing a behaviour rate $\tilde{d}_k \in [0, 1]$, assuming the constraints are binding and a feasible policy exists.

Indicator costs offer a practical advantage: they allow us to model both desired and undesired behaviours without affecting episode termination tendencies. With arbitrary cost functions, it may seem sufficient to flip their sign to enforce the opposite behaviour. However, as Kostrikov et al. [256] observed, whether a behaviour is enforced through bonuses or penalties interacts subtly with termination conditions: bonuses may encourage the agent to delay termination to accumulate more reward, while penalties may lead it to terminate prematurely. Indicator costs circumvent this issue: the opposite behaviour is enforced simply by inverting the indicator, $1 - I(s, a)$, without altering the sign of the constraint.

7.4.2 Multiplier Normalization

When a constraint k is violated, the corresponding Lagrange multiplier λ_k increases in order to place greater emphasis on satisfying that specific behavioural requirement. While it is crucial that the multipliers have enough capacity to grow relative to the main objective, constraints that are *slow to learn* can lead to multipliers reaching excessively large values. This, in turn, may produce very large policy updates and destabilize the overall learning dynamics. We show an example of this in our experiments in Section D.2.3

To preserve the ability of a single constraint to dominate the updates when necessary, while ensuring the overall scale of the updates remains bounded, we propose to *normalize* the multipliers. A simple and effective way to achieve this is through a softmax parametrization:

$$\lambda_k = \frac{\exp(z_k)}{\exp(a_0) + \sum_{k'=1}^K \exp(z_{k'})}, \quad k = 1, \dots, K, \quad (7.12)$$

where z_k are unconstrained parameters associated with each multiplier and a_0 is a dummy parameter used to allocate a normalized weight

$$\lambda_0 := 1 - \sum_{k=1}^K \lambda_k$$

to the main objective $J_R(\pi)$. With this reparametrization, the resulting min-max optimization problem becomes

$$\max_{\pi} \min_{z_{1:K} \geq 0} \mathcal{L}(\pi, \lambda) \quad \text{with} \quad \mathcal{L}(\pi, \lambda) = \lambda_0 J_R(\pi) - \sum_{k=1}^K \lambda_k (J_{C_k}(\pi) - d_k), \quad (7.13)$$

which ensures that all multipliers remain properly scaled, thereby stabilizing policy updates while still allowing individual constraints to take precedence when required.

7.4.3 Constrained GRPO

Following our proposed framework for behaviour specification, in this section we extend it to Group Relative Policy Optimization (GRPO). Recall that GRPO [66] estimates advantages by *within-group standardization*, i.e., subtracting the group mean and dividing by the group standard deviation (Section 2.3.2). We repeat the equation of the advantage estimation here for clarity of presentation,

$$\hat{A}_{i,t} = \frac{r_{i,t} - \text{mean}(\mathbf{r}_t)}{\text{std}(\mathbf{r}_t)} \quad (7.14)$$

In the presence of multiple objectives, there are two natural ways to linearly combine them within GRPO’s advantage estimation, and each has distinct implications for the resulting optimization dynamics.

Scalarized Rewards

In the *Scalarized Rewards* approach, we first collapse the main reward and all constraint costs into a single scalar by linear combination, and then apply GRPO’s within-group standardization to this combined signal (as is common in prior work [182]). The following result characterizes the resulting advantage in terms of per-component standardized variables.

Theorem 7.4.1 (Implicit reweighting under scalarized rewards). *Let*

$$\mathbf{x} := \begin{bmatrix} R \\ C_1 \\ \vdots \\ C_K \end{bmatrix} \in \mathbb{R}^{K+1}, \quad \boldsymbol{\lambda} := \begin{bmatrix} \lambda_R \\ \lambda_1 \\ \vdots \\ \lambda_K \end{bmatrix} \in \mathbb{R}^{K+1},$$

and define the scalarized reward $R_S := \boldsymbol{\lambda}^\top \mathbf{x}$. Denote the within-group mean and covariance of $\mathbf{x}$ by $\boldsymbol{\mu} := \mathbb{E}[\mathbf{x}]$ and $\boldsymbol{\Sigma} := \text{Cov}(\mathbf{x})$, respectively, and let

$$\sigma_{R_S} := \sqrt{\text{Var}(R_S)} = \sqrt{\boldsymbol{\lambda}^\top \boldsymbol{\Sigma} \boldsymbol{\lambda}}.$$

Further, define per-component standardizations

$$\begin{aligned} \mathbf{Z} &:= \mathbf{D}^{-1}(\mathbf{x} - \boldsymbol{\mu}), \quad \mathbf{D} := \text{diag}(\sigma_R, \sigma_{C_1}, \dots, \sigma_{C_K}) \\ \sigma_R &> 0, \quad \sigma_{C_k} > 0 \quad \forall k \in \{1, \dots, K\}, \end{aligned}$$

so that

$$\begin{aligned} Z_R &:= \frac{R - \mu_R}{\sigma_R}, \\ Z_{C_k} &:= \frac{C_k - \mu_{C_k}}{\sigma_{C_k}}, \quad k = 1, \dots, K, \end{aligned} \quad (7.15)$$

where $\mu_{(\cdot)}$ and $\sigma_{(\cdot)}$ denote the within-group mean and standard deviation of the corresponding component. Then the GRPO advantage obtained by standardizing the scalarized reward,

$$A_{\text{ScRew}} := \frac{R_S - \mathbb{E}[R_S]}{\sigma_{R_S}},$$

admits the expansion

$$A_{\text{ScRew}} = \frac{(\mathbf{D}\boldsymbol{\lambda})^\top \mathbf{Z}}{\sigma_{R_S}} = \sum_{j=0}^K \left(\frac{\lambda_j \sigma_j}{\sigma_{R_S}} \right) Z_j, \quad (7.16)$$

where we index $j = 0$ as the main reward component ($\lambda_0 \equiv \lambda_R$, $\sigma_0 \equiv \sigma_R$) and $j = k$ corresponds to cost C_k . Consequently, the effective contribution of each standardized component Z_j is scaled by $\lambda_j \sigma_j / \sigma_{R_S}$, and therefore depends on within-group scales and (through σ_{R_S}) the full covariance structure $\boldsymbol{\Sigma}$.

Proof. By definition, $R_S = \boldsymbol{\lambda}^\top \mathbf{x}$ and $\mathbb{E}[R_S] = \boldsymbol{\lambda}^\top \boldsymbol{\mu}$, hence

$$R_S - \mathbb{E}[R_S] = \boldsymbol{\lambda}^\top (\mathbf{x} - \boldsymbol{\mu}).$$

Moreover,

$$\text{Var}(R_S) = \text{Var}(\boldsymbol{\lambda}^\top \mathbf{x}) = \boldsymbol{\lambda}^\top \text{Cov}(\mathbf{x}) \boldsymbol{\lambda} = \boldsymbol{\lambda}^\top \boldsymbol{\Sigma} \boldsymbol{\lambda},$$

which yields $\sigma_{R_S} = \sqrt{\boldsymbol{\lambda}^\top \boldsymbol{\Sigma} \boldsymbol{\lambda}}$.

Next, from the definition of $\mathbf{Z}$, we have $\mathbf{x} - \boldsymbol{\mu} = \mathbf{D}\mathbf{Z}$. Substituting into Eq. (7.4.3) gives

$$R_S - \mathbb{E}[R_S] = \boldsymbol{\lambda}^\top \mathbf{D}\mathbf{Z} = (\mathbf{D}\boldsymbol{\lambda})^\top \mathbf{Z}.$$

Dividing both sides by σ_{R_S} yields

$$A_{\text{ScRew}} = \frac{R_S - \mathbb{E}[R_S]}{\sigma_{R_S}} = \frac{(\mathbf{D}\boldsymbol{\lambda})^\top \mathbf{Z}}{\sigma_{R_S}},$$

which is the first equality in Eq. (7.16). Expanding the dot product gives the summation form in the second equality, completing the proof. $\square$

To summarize, our **main observation** is revealed in Eq. (7.16): under scalarized rewards,

the effective contribution of each standardized component Z_j and the intended trade-offs encoded by the Lagrangian multipliers, λ , are implicitly distorted by component variances and correlations.

Scalarized Advantage

In our constrained GRPO approach, we first compute a GRPO-style standardized advantage *per component* (reward and each constraint) using the respective component’s within-group mean and standard deviation as in Eq. (7.15). We then form a single advantage by linearly combining these standardized components using their current Lagrangian weights

$$A_{\text{ScAdv}} = \lambda_R Z_R - \sum_{k=1}^K \lambda_k Z_{C_k}, \quad (7.17)$$

where $\lambda_k, \lambda_R \geq 0$ are Lagrange multipliers associated with constraint k and the reward term, respectively. This yields an advantage signal whose relative trade-offs are explicitly controlled by the multipliers.

Main Takeaway

We conclude that if one wants λ to faithfully encode their intended reward-constraint trade-off independently of group scales and covariance, the per-component standardization followed by combining terms using λ is the more faithful approach. In general, scalarizing rewards is not invalid but has the caveat that it implicitly reweighs the multipliers by the components’ standard deviations (and ρ). This results in agents effectively having to satisfy the high-variance components first prior to satisfying the low-variance ones, *regardless of the Lagrangian multiplier λ_i values*. We illustrate the differences with a numerical example in Appendix D.4.1. In our experiments in Section 7.5.2, we present results on a toy environment and in the real-world NAVSIM driving benchmark, comparing both approaches with fixed and varying Lagrangian multipliers.

7.5 Experiments

To evaluate our proposed constrained-GRPO, we first present experiments on a toy gridworld environment to showcase the empirical difference between *scalarized rewards and scalarized advantages*. Finally, we evaluate our approach in the realm of large vision language models for autonomous driving with GRPO [66] finetuning. In this setting, the agent is an autonomous driving vehicle tasked with generating a plan through a driving scene as shown in Figure 7.2.

The success of GRPO with specified constraints on dealing with long-tail scenes highlights the applicability of this framework to real-world problems. In Appendix D, we further validate our proposed framework with a constrained version of soft Actor-Critic [59] on a set of constrained tasks for deploying RL agents in video games.

7.5.1 Constrained GRPO: Gridworld Environment

We start our experiments with an intentionally minimal gridworld environment as it allows us to isolate the differences between *scalarized rewards* and *scalarized advantages* in unconstrained and constrained GRPO. In this environment, an agent is randomly initialized on a 10×10 grid and is tasked with navigating to a randomly sampled goal location, receiving a sparse scalar reward only upon reaching the goal. Each episode contains 20% lava tiles, and stepping onto lava incurs a penalty (i.e., a cost). In addition, the agent has a battery that depletes at every timestep and receives a penalty whenever its battery level drops below 10%. The action space comprises five discrete actions: moving in the four cardinal directions, or staying in place to recharge. Further environment details are provided in Appendix D.4.4.

We begin by comparing the two scalarization strategies under fixed reward and penalty weights. We consider two settings for the battery-penalty weight, $\lambda_{\text{battery}}^{\text{fixed}} \in 0.0, 0.1$, and sweep the lava-penalty weight over $\lambda_{\text{lava}}^{\text{fixed}} \in 0.0, 0.01, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0$. Figure 7.5 reports performance as a function of $\lambda_{\text{lava}}^{\text{fixed}}$ for each choice of $\lambda_{\text{battery}}^{\text{fixed}}$. Across the sweep, *scalarized advantages* achieves consistently higher goal-reaching rates and exhibits greater seed-level stability (smaller error bars). By contrast, *scalarized rewards* tends to optimize the lava rate in a unilateral manner: even the smallest non-zero lava weight, $\lambda_{\text{lava}}^{\text{fixed}} = 0.01$, drives lava visitation to near-minimal levels. *Scalarized advantages* instead yields smoother, more gradual control of the lava rate, which we argue better matches the intended semantics of the weights. When both penalties are active ($\lambda_{\text{battery}}^{\text{fixed}} = 0.1$, bottom row of Fig. 7.5), *scalarized advantages* relaxes the lava rate for $\lambda_{\text{lava}}^{\text{fixed}} \leq 0.5$, whereas *scalarized rewards* behaves essentially identically to the $\lambda_{\text{battery}}^{\text{fixed}} = 0.0$ case—effectively optimizing the lava penalty the same way regardless of whether the battery penalty is present.

We next evaluate the two scalarization strategies in the constrained regime. In this experiment, we enforce thresholds $\tilde{d}_{\text{lava}} = \tilde{d}_{\text{battery}} = 0.01$ and train for 512k episodes; Appendix D.4.4 reports additional results under different threshold values. Figure 7.6 compares Constrained GRPO using *scalarized advantages* (ours) to a *scalarized rewards* baseline. In line with the fixed-weight setting, *scalarized advantages* produces more stable learning across five seeds, evidenced by the consistently narrower shaded region throughout training (row 1). It also yields richer behavior-rate dynamics (row 2,3, left column): the Lagrangian multipliers

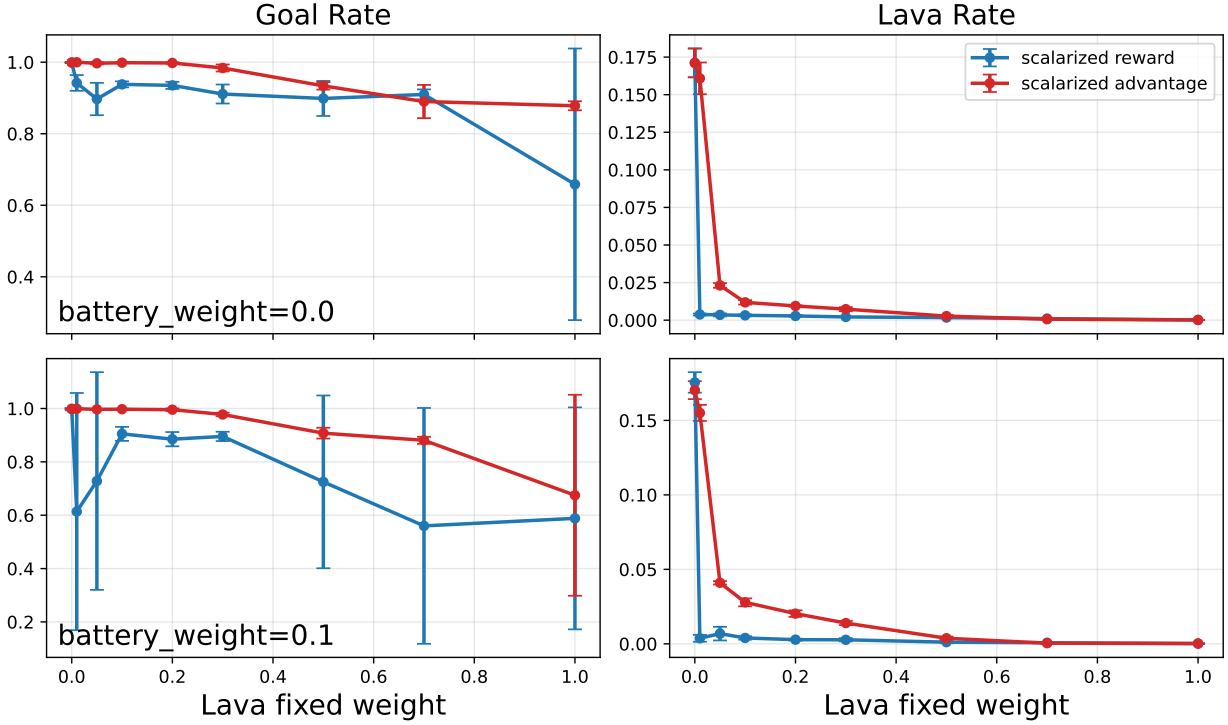


Figure 7.5 **Fixed-weights GRPO comparing Scalarized Rewards with Scalarized Advantages.** Final performance of GRPO policies evaluated over 1,000 episodes in the gridworld with fixed penalty weights, comparing *scalarized rewards* (blue) and *scalarized advantages* (red). Scalarized advantages consistently achieves higher goal-reaching rates while tracking the intended trade-off induced by the lava-weight sweep more faithfully, whereas scalarized rewards suppresses lava visitation disproportionately.

(row 4, left column) adapt so that the constraint rates rise toward the threshold, effectively encouraging the policy to use its available cost budget. In contrast, *scalarized rewards* quickly drives both constraint rates near zero early in training (row 2,3, right column) and keeps them there, even as the learned multipliers remain small (row 4, right column).

This behavior is explained by the implicit reweighting described in Theorem 7.4.1. In particular, we call the scaling factors applied to the Z_j terms in Eqs. (7.16) and (7.17) the *effective weights*. Notably, even though the learned multipliers λ_{lava} and λ_{battery} remain small, the resulting effective weights are both larger and substantially noisier due to within-group variance and covariance effects (row 5, right column). This becomes especially problematic when the induced importance ordering shifts, for example, around 150K steps where the goal and lava weights swap (red/green). Overall, these findings support *scalarized advantages* as a more dependable advantage construction for Constrained GRPO, since it preserves the intended role of the Lagrangian multipliers in enforcing behavior specifications.

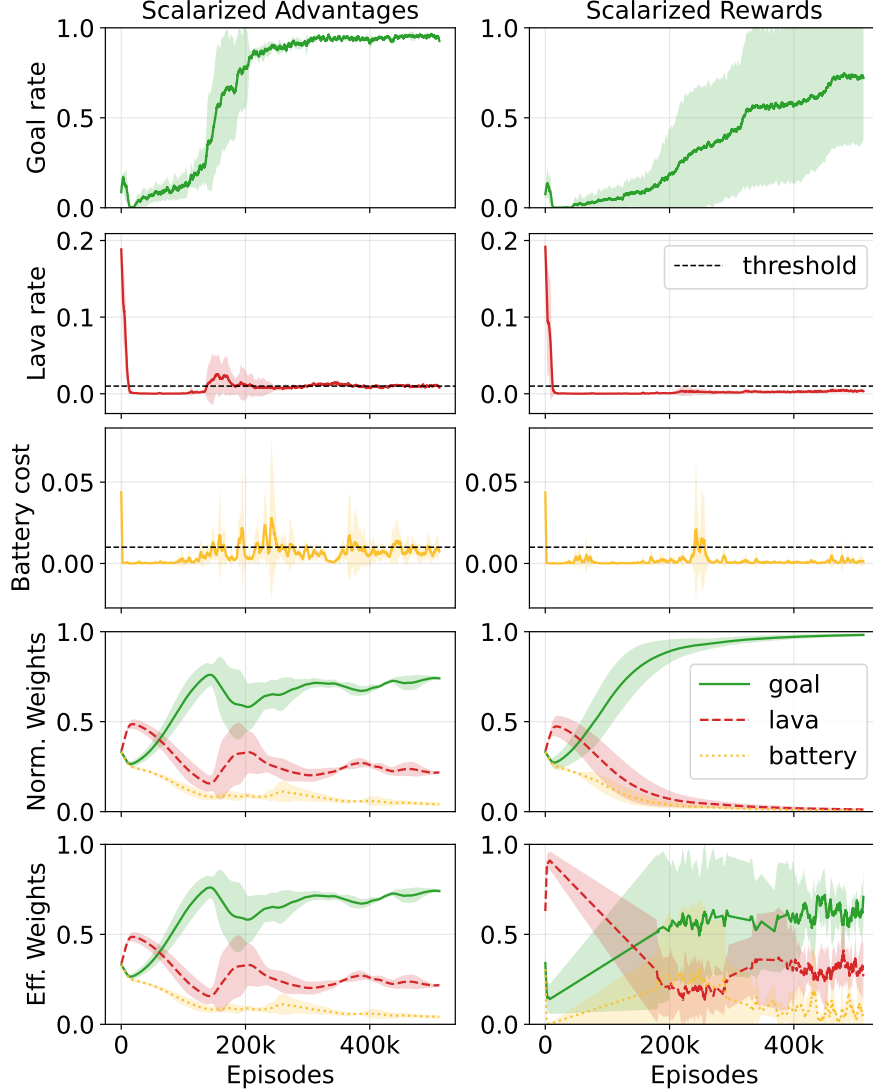


Figure 7.6 **Learning Dynamics of Constrained GRPO with $\tilde{d}_{\text{lava}} = 0.01$ and $\tilde{d}_{\text{battery}} = 0.01$** on the gridworld, comparing *scalarized advantages* (left) and *scalarized rewards* (right). Scalarized advantages yields more stable training and uses the available cost budget, with behavior rates adapting toward the specified thresholds as the Lagrangian multipliers adjust. In contrast, scalarized rewards collapses constraint rates toward near-zero early in training with the effective weights remaining larger and noisier despite small learned multipliers.

7.5.2 Constrained GRPO: Autonomous Driving Benchmark

Benchmark Details We focus our experiments on the NAVSIM (v2) benchmark [257] which consists of 103k training (Navtrain) and 12k testing (Navtest) scenarios curated from OpenScene [258], a redistribution of the nuPlan [6] dataset. The NAVSIM v2 benchmark proposes to evaluate agents using pseudo-simulation on the most difficult scenarios from

NAVSIM challenge [259] (Navhard split) with a two-stage evaluation protocol. The first stage introduces simulation metrics to evaluate the closed-loop planning capability of planners in an open-loop evaluation manner. Specifically, the 4-second generated trajectory (sampled at 10-hz) is executed without feedback in the world using the kinematics bicycle model with an LQR controller. While the first version of Navsim [259] executed the recorded trajectories without reacting to the ego agent, Navsim-v2 employs rule-based reactive planner, called Intelligent Driver Model (IDM;[32]) to improve the simulation realism. The second stage adopts Multi-Traversal Gaussian Splatting [260] to synthesize subsequent driving scenarios which start from sub-optimal position and/or heading. The resulting Navhard evaluation split consists of in 450 stage 1 and 5,462 stage 2 observations.

The primary metric introduced in the NAVSIM-v2 benchmark is the Extended PDM score (EPDMS), named after the rule-based planner Predictive Driver Model (PDM; [261]). The metric is calculated by aggregating multiple scores together using the following:

$$\text{EPDMS} = \underbrace{\prod_{m \in \mathcal{M}_{\text{pen}}} \text{filter}_m(\text{agent}, \text{human})}_{\text{penalty terms}} \cdot \underbrace{\left(\frac{\sum_{m \in \mathcal{M}_{\text{avg}}} w_m \text{filter}_m(\text{agent}, \text{human})}{\sum_{m \in \mathcal{M}_{\text{avg}}} w_m} \right)}_{\text{weighted average terms}} \quad (7.18)$$

The *penalty terms* consist of (1) no at-fault collisions, (2) driving within the drivable area, (3) driving in the correct direction, and (4) respecting traffic light signalization. As we can see from Equation 7.18, if any of these penalty terms are 0 (i.e., due to a violation of that term), the planner receives a score of 0 for that scenario. The filter_m function is a filtering mechanism on the penalty terms which ensures that if a rule is violated by the human expert driver in that scenario, the penalty violation gets ignored. The *weighted average terms* consists of (1) the ego-progress measured along the route ($w_m = 5$), (2) time-to-collision to ensure safety margins with other agents are respected ($w_m = 5$), (3) history comfort ($w_m = 2$), (4) extended comfort ($w_m = 2$), which evaluate the comfort of the plan in relation to the history and future, and (5) lane keeping ($w_m = 2$). The final EPDM score is a scalar value between $[0, 1]$. We also report the average displacement error (ADE) at 4 seconds of the output trajectory compared to the ground-truth for stage one scenarios.

Base Model Pretraining

We start from a pre-trained VLM, a Qwen2.5-VL 3B Instruct model [262], that we first supervise finetune on Covla dataset [263], a dataset of 83 hours of nominal Japanese driving. Specifically, we follow the initial pre-training recipe highlighted in [264], where we treat generating a planned trajectory as a next-token prediction problem in the original token

vocabulary, thus representing trajectories as text. Furthermore, the model is also trained with automatically-generated chain-of-thought annotations by a pre-trained Qwen2.5-VL 72B Instruct in a zero-shot manner, and concatenated to the trajectory strings. Following this first stage of supervised finetuning, we further supervise finetune the model on 92,000 scenes from the Navsim benchmark [257, 259], leaving $\approx 11,000$ scenes for RL finetuning. This allows the model to learn domain-specific driving for the benchmark.

Model Configurations

Recall that we are primarily interested in the usefulness of formulating behaviour specification as a constrained RL problem. We implement a set of baselines to showcase the performance of representative methods proposed in the literature. All of our experiments in this Section employ GRPO as the RL algorithm. Below we present all the variants we compare in our experiments.

1. EPDMS as rewards: This approach applies the EPDMS metric as the reward function. CaRL [265] and Plan-R1 [184] both employed a similar reward function; CaRL in a closed loop setting with PPO as the learning algorithm while Plan-R1 in the nuPlan benchmark. The authors propose the use of this reward function since they found that multiple reward functions would result in exploitative behaviour by the agent. We can view this approach as imposing a hard constraint on behaviour when the agent commits *any* violations.
2. Equally-weighted linear terms: Perform a linear combination of penalty components and the main weighted average terms (WAT) with equal weights set to $w_i = 0.25$:

$$R(\tau) = w_1\text{NC} + w_2\text{DAC} + w_3\text{DDC} + w_4\text{WAT} \quad (7.19)$$

where WAT corresponds to the weighted average terms in Equation 7.18. This is the approach employed by prior work [182, 183]. While Li et al. [182] scalarize the rewards (see Section 7.4.3) while Ichihara et al. [183] scalarize the advantages (see Section 7.4.3).

3. Informed weight terms: Given our experience with the dataset from initial trial-and-error experimentation, we attempt to manually set the weights in order. These were informed by the results of the SFT model. We set the weights to $w_1 = 0.1, w_2 = 0.8, w_3 = 0.05, w_4 = 0.05$ in Equation 7.19. We evaluate both scalarization approaches as in 2.

4. **Constrained GRPO:** The last model variant is the constrained GRPO version of Equation 7.19. That is, we aim to maximize the main reward (i.e., WAT) subject to minimizing violation of the constraints: NC, DAC and DDC. For simplicity, in this section, we ignore the bootstrap constraint as there is no clear notion of a success in our task since we do not have a goal position. All constraint satisfaction thresholds are set to 0.99. We evaluate the less optimal version with scalarized rewards, and our proposed fix to constrained GRPO with scalarized advantages

We found that *traffic light compliance (TLC)* was almost always respected; we chose to ignore it in almost all baselines, with the exception of EPDMS.

Results

We finetune all our models for two epochs through the 11k held-out scenes for RL finetuning. In our experiments, the agent is limited to only observing the three front views in Figure 7.2, due to computational limitations, along with its speed and intention as textual prompts. All our experiments are performed on 8 A100 GPUs, with each RFT run taking nearly 20 hours. We set the beta parameter to 0.02 to constrain the policy to the reference SFT policy, a learning rate of $1e-6$, set the number of group generations to 16, and perform two iterations of policy optimization per batch of data. We update the Lagrangian multipliers at the same rate as the policy. Tables 7.1 and 7.2 present the results of all the baselines on NAVSIM Stage 1 and Stage 2, respectively. We add the PDM [261] planner’s results, the planner whose predictions are used to compute the extended PDM score for the agent’s trajectories. In the following, we present our observations about our results.

Behaviour specification through constrained RL is useful for AV planners. Our results align with our prior belief that AV planners stand to benefit from the behaviour specification through constrained RL framework. Specifically, we observe that our constrained GRPO variants (bottom two rows in Tables 7.1 and 7.2) produce the best overall results, with the scalarizing advantages approach having the highest EPDMS. Specifically, for in-distribution (stage 1) scenarios, both models provide a significant improvement in performance compared to the starting SFT checkpoint; this is not true of other baselines. For out-of-distribution (stage 2) scenarios, as expected, there is a performance drop across all metrics. However, both constrained GRPO models still exhibit strong performance, showcasing the usefulness of our behaviour specification framework for limiting bad driving behaviours.

Configuration Name	NC $\uparrow$	DAC $\uparrow$	DDC $\uparrow$	TLC $\uparrow$	Ego Progress $\uparrow$	Comfort $\uparrow$	EPDMS $\uparrow$	ADE@4s (m) $\downarrow$
PDM-C	0.944	0.988	1.00	0.995	1.00	0.877	-	-
SFT Checkpoint	0.953	0.789	0.978	0.993	0.836	0.976	0.651	2.539
GRPO with EPDMS [184, 265]	0.897	0.800	0.947	0.990	0.944	0.978	0.612	6.900
GRPO, ScRew, Equal Weights [182]	0.891	0.625	0.876	0.989	0.968	0.976	0.445	9.050
GRPO, ScAdv, Equal Weights [183]	0.908	0.661	0.900	0.989	0.954	0.973	0.490	7.172
GRPO, ScRew, Informed Weights	0.887	0.643	0.887	0.987	0.967	0.978	0.463	8.509
GRPO, ScAdv, Informed Weights	0.932	0.765	0.971	0.993	0.890	0.976	0.623	3.495
Constrained GRPO, ScRew	0.955	0.887	0.987	0.993	0.868	0.976	0.743	3.344
Constrained GRPO, ScAdv (ours)	0.957	0.928	0.989	0.998	0.796	0.977	0.774	3.361

Table 7.1 **Stage 1 NAVSIM-v2 Results.** Stage 1 results on the NAVSIM-v2 benchmark’s Navhard split which measures the policy’s ability to generalize to unseen scenarios during training. We compare the different reward function configurations explored in the literature and our proposed Constrained GRPO approach, as depicted in Section 7.5.2. *ScRew* stands for Scalarized rewards, while *ScAdv* stands for Scalarized Advantages. We can see that the best performing models are the constrained approaches, with our scalarized advantages approach to constraining behaviour performing the best on the EPDMS metric.

Scalarized advantages outperform scalarizing rewards. Our results reveal the clear pattern that scalarizing advantages provides a significant improvement in performance compared to scalarizing rewards. Specifically, across all three weight variants (equal, informed and constrained) focusing only on EPDMS, we see that scalarizing advantages *always* outperforms scalarizing rewards; this is true for both the in-distribution and out-of-distribution scenarios. Our mathematical analysis in Section 7.4.3 provides a hint for why this is: not accounting for the variances of different components (rewards or constraints) hurts the optimization process since the advantage, and therefore the gradient with respect to the parameters, will be dominated by components with higher variances.

To further understand the differences, we bring the reader’s attention to Figure 7.7. We can see that the constrained GRPO with scalarized rewards does not seem to react to the Lagrangian multiplier weight adjustments. This is particularly obvious in the drivable area compliance (DAC) and the ego-progress (EP), which is a proxy for the main reward WAT. Specifically, even though the weight for DAC is near its maximum of 1.0 and the reward weight is near 0.0, the DAC constraint never reaches a satisfactory level, while the ego progress goes up, and exhibits a very slow decay until the end of training. In contrast, constrained GRPO with scalarized advantages reacts within the first 1000 steps of training to the setting of the DAC Lagrangian multiplier. Furthermore, we see that the ego-progress diminishes in reaction to its multiplier decreasing to near 0.0. Finally, we observe an interesting behaviour in the multipliers with the constrained advantages variant at around 2500 steps, where the multiplier weight for the no at-fault collisions (NC) constraint rises sharply. This occurs because, at that training step, DAC had reached a satisfactory level, whereas the NC constraint dropped

sharply. This confirms our mathematical findings that scalarizing advantages ensures the multipliers have their intended effect.

Configuration Name	NC $\uparrow$	DAC $\uparrow$	DDC $\uparrow$	TLC $\uparrow$	Ego Progress $\uparrow$	Comfort $\uparrow$	EPDMS $\uparrow$
PDM-C	0.881	0.906	0.963	0.985	1.00	0.915	-
SFT Checkpoint	0.800	0.756	0.869	0.981	0.878	0.963	0.444
GRPO with EPDMS [184, 265]	0.749	0.759	0.829	0.969	0.979	0.945	0.381
GRPO, ScRew, Equal Weights [182]	0.751	0.696	0.787	0.968	0.986	0.954	0.347
GRPO, ScAdv, Equal Weights [183]	0.767	0.705	0.813	0.971	0.981	0.949	0.370
GRPO, ScRew, Informed Weights	0.754	0.695	0.791	0.969	0.986	0.954	0.350
GRPO, ScAdv, Informed Weights	0.778	0.731	0.831	0.972	0.944	0.962	0.385
Constrained GRPO, ScRew	0.810	0.784	0.866	0.976	0.939	0.952	0.436
Constrained GRPO, ScAdv (ours)	0.853	0.830	0.897	0.986	0.863	0.967	0.516

Table 7.2 **Stage 2 NAVSIM-v2 Results.** We present Stage 2 results on the NAVSIM-v2 benchmark’s Navhard split which measures the policy’s ability to generalize to long-tail settings where agents need to produce corrective behaviours. We compare the different reward function configurations explored in the literature and our proposed Constrained GRPO approach, as depicted in Section 7.5.2. *ScRew* stands for Scalarized rewards, while *ScAdv* stands for Scalarized Advantages. We see significant improvement in the EPDMS score with our constrained GRPO with scalarized advantages approach. This indicates the scalarized advantages allow the model to learn the correct signal for driving as it is less susceptible to the noise of the constraint terms.

Hard constraints significantly impact the learning dynamics. Our experiments indicate that employing a hard-constraint approach (i.e., *GRPO with EPDMS* [265]) produces suboptimal behaviours compared to a soft-constraint approach. This is true for both the in-distribution (stage 1) and out-of-distribution generalization (stage 2) as shown by the overall EPDMS metric being lower. As our analysis suggested in Section 7.4.3, when different signals constitute the reward function, it is important to account for different variances of individual terms. The hard-constraint approach does not allow us to make this distinction, which we believe makes the learning problem more difficult.

7.6 Conclusion

Our work demonstrates that CMDPs provide a powerful and flexible framework for task specification in reinforcement learning. In particular, we proposed a framework that expresses desired agent behaviours as constraints on the frequency of specific indicator events within a CMDP formulation. Through a series of experiments, we showed that this constraint-based methodology is preferable to traditional reward engineering, which typically requires extensive hyperparameter searches over candidate reward functions. We evaluated our framework

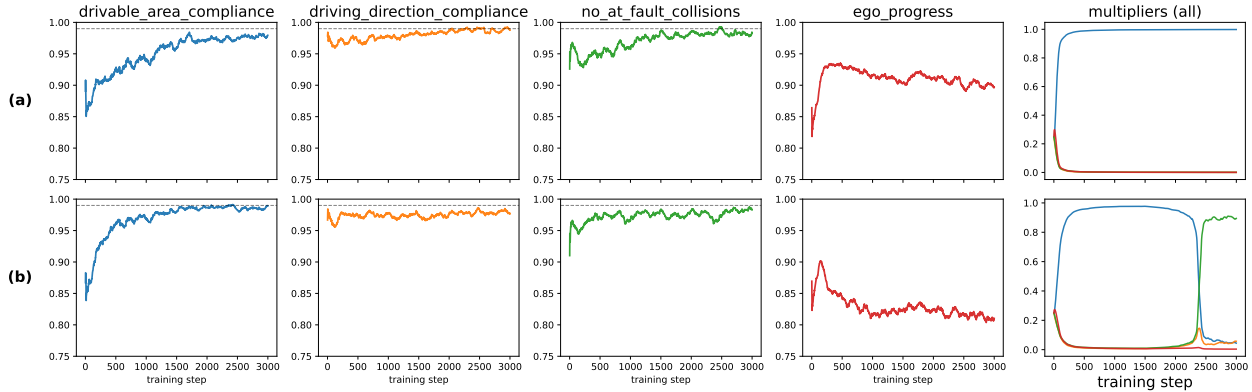


Figure 7.7 Multiple Constraints on NAVSIM training. Constrained GRPO plots of constraint and multipliers values through training, comparing with (a) scalarized rewards with (b) scalarized advantages. Focusing on the ego progress and the drivable area compliance, we see that with the scalarized rewards, the policy does not adhere to the multiplier settings as we intend. First, the drivable area compliance (DAC) stays below its minimum threshold 0.99 throughout its training even though the multiplier value had moved close to its maximum value of 1.0. Moreover, we see that even though the ego-progress (a component of the WAT reward) is set to a very low value, the scalarized rewards variant still maximizes it. Conversely, the scalarized advantage model allows the different components to adhere to the multiplier setting. In addition, we see that at around 2500 training steps, something interesting happens in the scalarized advantage model’s multipliers; since the DAC constraint is satisfied and the no at-fault collision (NC) takes a sharp dip, the Lagrangian optimization pushes the NC multiplier higher and squashes the DAC multiplier. This confirms our theoretical findings that scalarizing the advantages leads to better adherence to the specified multiplier weights.

in scenarios involving multiple simultaneous constraints across three different environments. Our results indicate that satisfying a large number of constraints simultaneously is inherently challenging and can systematically hinder progress on the main task of interest. To address this, we introduced a normalization scheme for the constraint multipliers, which improved training stability, and proposed a *bootstrap constraint* to guide the early stages of learning. This bootstrap mechanism allows practitioners to incorporate prior knowledge about the task: when the success threshold is stringent, achieving high task performance is prioritized, while a more relaxed threshold simply helps the agent escape the initialization regime, after which the other constraints dominate. Overall, as we have shown, our method is simple to integrate into existing policy-gradient implementations and scales effectively across domains. We extended our framework to the RL finetuning of large vision-language models with GRPO and showed that scalarizing advantages is the principled way to incorporate constraints into the GRPO learning objective. We hope that these insights will encourage broader adop-

tion of constrained reinforcement learning methods in autonomous vehicle settings and foster stronger interactions between the research and applied RL communities.

CHAPTER 8 CONCLUSION

Autonomous driving is a grand challenge at the intersection of robotics, computer vision, and artificial intelligence, aiming for safe and efficient navigation without humans. Solving it promises major gains in transportation safety, efficiency, and accessibility. Modern autonomy stacks are modular, with perception, prediction, planning, and control modules that communicate sequentially. The perception stack builds an environment representation from sensors while the remaining modules decide what the autonomous vehicle should do. This thesis sets out to advance the state of knowledge of data-driven approaches for behaviour prediction, planning and multi-agent behaviour simulation.

8.1 Synthèse des travaux / Summary of Works

In Chapter 4, we introduced Multi-Agent Motion Forecasting with Sequential Set Transformers. This first contribution focused on multi-agent motion forecasting, where we cast this problem as a sequential set prediction problem and introduced AutoBots, latent-variable encoder-decoder transformers. Our use of discrete latent seed parameters provides an inference-efficient way to model distinct behavioural modes for agents, while the factorized attention mechanism provides a nice way to capture inter-agent interaction and scene context. Through extensive experiments, we showed that our model is both effective and accurate at modelling the uncertain future trajectory of ego-agents, and can generate scene-consistent multi-agent scenarios.

In Chapter 5, we introduced CtRL-Sim, a framework for multi-agent controllable simulation using return-conditioned supervised learning. CtRL-Sim is trained to imitate policies from logged driving data using a next-token prediction transformer over actions, return-to-gos, and state rollouts, executed in a physics-aware Nocturne [38] simulator. We show that the resulting agents are realistic, diverse, and crucially *controllable* at inference via exponential tilting [40], which adjusts behavior rates without retraining. With this work, we aim to address two important problems in behaviour simulation: (1) replacing non-reactive log replay agents with reactive ones; and (2) generating long-tail adversarial scenarios to evaluate planners.

In Chapter 6, we focus our attention on initial scene generation in the abstracted world using our proposed vectorized latent diffusion model. We show this approach is an efficient and effective architectural prior compared to rasterizing the vectorized scene and employing an

image-based diffusion model. Combining our vectorized latent diffusion model with CtRL-Sim, we propose a fully data-driven simulation framework trained on driving log data. We evaluate self-play trained PPO policies on scenario-dreamer generated long-tail scenarios, showcasing its possible future usage for planner evaluation.

In Chapter 7, we propose the use of constraint-satisfying policies in reinforcement learning. Specifically, we formalized the direct behaviour specification problem via Constrained Markov Decision Processes. We showcased how traditional scalarized rewards with fixed weights are a limiting approach, and does not scale well with the number of reward functions being combined. With our proposed framework, designers can choose measurable upper bounds on undesired behavior (e.g., collision or off-road rates) while retaining task-efficiency objectives. The formulation proved effective in interactive video-game domains, even with a large number of competing constraints. We extend our framework to the special case of the recently-proposed Group Relative Policy Optimization (GRPO) and show the importance of scalarizing advantages for the framework’s effectiveness. With this, we show the scalability of constrained-GRPO in the context of RL fine-tuning of large Vision-Language models. This illustrates that constraint-aware optimization can guide even powerful function approximators toward acceptable policies without hand-tuned reward shaping.

8.2 Limitations de la solution proposée / Limitations

While we believe our work in this thesis has advanced the state of knowledge for behaviour modelling in autonomous driving, we believe there are several limitations.

- **Scene Abstraction.** By design, we focused our attention on post-perception vectorized worlds in all of our proposed work. While this is fine for current modular stacks, it is the author’s belief that future work will move more and more towards end-to-end driving. As such, it remains to be seen if the methods explored in this thesis would easily scale to domains where the input is raw sensor inputs. We believe that since the publication of our work in Chapter 4, it has been shown that the central idea of AutoBots, namely factorized attention for spatial and temporal axes, extends well to the domain of end-to-end driving, i.e., starting with sensor data [107]. Our experiments in Chapter 7 on VLMs for autonomous driving showcase the extendibility of behaviour specification through constrained RL to sensor inputs. However, the problem of simulating sensor-data in multi-agent environments remains an open problem [266].
- **Distribution Shifts in behaviour simulation** Our behaviour simulation framework, presented in Chapter 5 and employed in Chapter 6, employs offline reinforcement learn-

ing to simulate long-tail behaviours. While we showed this approach is effective at challenging planners, we observed a failure case: negatively tilting agents produced adversarial agent behaviours where agents looked to crash with other agents. We see this as a direct consequence of the formulation of the reward function.

- **Controllability calibration.** While CtRL-Sim from Chapter 5 employed exponential tilting knobs for controllability, it was hard to decide exactly what value of the tilting coefficients one should use. We believe control knobs for effective simulation frameworks should have clear indicators of the knob values to what the resulting behaviours will produce.
- **Large-Scale Simulation.** To support real autonomous driving planner training and evaluation, we need large realistic worlds with realistic, closed-loop, reactive agent behaviours. While this thesis presented solutions towards this end goal in Chapter 6, even in our abstracted vectorized worlds, we still cannot effectively generate large coherent maps. In addition, in such large worlds we need agents that can continuously drive in these large environments. While CtRL-Sim can be effective for the 9 seconds scenes from Waymo dataset [223], it is unclear how it would fare in scenes that are 1-2 minutes long.
- **Scalability and compute.** As with the general field of machine learning, the problem of computational demands came more and more to the forefront during the development of this thesis. High-capacity transformers, diffusion models and large vision-language models at scale are computationally intensive. We avoided the problem of practical deployment of most of our algorithms as this requires further engineering for latency, memory, and energy budgets. AutoBot is the only model we had the chance to deploy in real-world autonomous driving setting with Torc Robotics. Academic research will need to address the problem of scarcity of compute compared to industrial labs.

8.3 Context of the Work and Author Contributions

In this section, I provide contextual information on the publications featured in this thesis. In addition, I provide the specific contributions I made to each publication.

8.3.1 AutoBots

The work in Chapter 4 is joint work with Florian Golemo, Felipe Codevilla, Martin Weiss, Jim Aldon D’Souza, Samira Ebrahimi Kahou, Felix Heide and Christopher Pal. Through

many discussions with my co-authors, I contributed the core intuition around treating trajectories as sequential sets with transformers and introduced a latent-variable formulation to capture multi-modal predictions. The factorized attention over temporal and spatial dimensions organically came out of many discussions. I implemented most of the codebase, with Felipe Codevilla leading the nuScenes data setup and Florian Golemo supporting results aggregation. Martin Weiss helped further clarify the manuscript and wrote the proofs for equivariance of the encoder and decoder. Samira Ebrahimi Kahou, Felix Heide and Christopher Pal supervised the work throughout.

Context

The pandemic was in full effect when this project started and the collaboration with all my co-authors had been moved to the online sphere. We were initially tackling this problem from the RL perspective in driving, building a toy environment and using multi-head attention to perform cross-attention between agents. Once we decided to explore real-world datasets, such as nuScenes, we began our pursuit of novel architectures for dealing with spatial-temporal data; we later coined the result of our exploration AutoBot.

Recent Developments

AutoBots was one of the first methods to propose the powerful idea of factorized attention. It was concurrent work with the Scene Transformer [109]. Since then, Plenty of work has built on ideas from Autobots; here we focus on work in motion prediction for autonomous driving. Work that followed on Autobots and concurrent work included QCNet [267], Wayformer [268] which both used interleaved factored attention similar to Autobots. Further work, such as motionLM [222], approached the trajectory forecasting problem as an autoregressive transformer model; we also present a similar work in Chapter 5. Finally, other work took inspiration from Autobots to design a language-conditioned diffusion model for multi-agent simulation [36, 132] and even used vanilla diffusion models for joint motion forecasting using factorized attention [115]. In industry, we have seen AutoBots used and cited by autonomous driving companies including Waymo, Torc Robotics and Nvidia.

8.3.2 CtRL-Sim

The work in this chapter was published at CoRL 2024. This is joint work with Luke Rowe (equal Authorship), Anthony Gosselin, Florian Golemo, Bruno Carrez, Felix Heide, Liam Paull and Christopher Pal. For this work, I developed the main idea behind using reward

conditioned supervised learning with multiple reward components, and the use of exponential tilting for adversarial behaviour simulation. Luke Rowe developed the novel multi-agent transformer architecture as well as the main infrastructure to train simulation models. Anthony helped with idea generation, helping debug and code the simulator and running experiments. Bruno was responsible for writing the main code for the Box2d simulator, built on top of Nocturne [38], which supported evaluation in realistic environments. Florian, Felix, Liam and Chris all provided supervision for this project throughout, as well as helped with the writing and architectural Figure generation (mainly Florian).

Context

I initially started this project with the help of Bruno Carrez for the simulator (from March 2022 until January 2023) and Anthony Gosselin as a new Master’s student. Florian came in to help with the conceptualization, while Felix and Chris provided supervision. We were initially building the reward-conditioned model using the codebase for AutoBots, while the idea for using multiple reward functions came to me while working on the project in the previous Chapter (constrained RL). I believe Luke joined the project in the Summer 2023 and at which point I started getting busy with a few personal life matters. He started working with the AutoBots codebase and experienced a little bit of success but then transitioned to the new multi-agent transformer architecture.

Recent Developments

This work has been well received by the industry with teams from Ubisoft, Waymo, Torc and Fortelix interested in building on the work. Academically, the work has been cited 11 times as of this moment, with some work on superintelligent agent risks [269], and other works building visual versions of CtRL-Sim [266]. This work is the first to explore the strengths of exponential tilting and test-time behavior modification through the lens of reinforcement learning in the realm of autonomous driving.

8.3.3 Scenario Dreamer

In this chapter, we present our proposed Vectorized latent diffusion approach for generating driving environments. This work was done with Luke Rowe (First Author), Anthony Gosselin, Liam Paull, Christopher Pal and Felix Heide. I mainly operated in a secondary role in this project, contributing to discussions and the overall design of the Scenario Dreamer architecture and proposed overall approach for simulation. In addition, I trained the RL

planning agent in GPUdrive and adapted the GPUdrive codebase accordingly to make the trained RL agent compatible with the Scenario Dreamer representation. Furthermore, I helped set up the infrastructure on the Torc Robotics’ research servers to support the project 8-GPU nodes for training. Anthony helped with running experiments on the cluster, while Liam Paull, Chris Pal and Felix Heide provided valuable supervision.

Context

At the start of the Scenario Dreamer project, I was working as a full-time researcher at Torc Robotics. There, I was focused on behaviour simulation, prediction and planning for Torc’s autonomy stack. We invited Luke Rowe to come work as a research intern in the summer of 2024 at Torc Robotics following our joint work on CtRL-Sim (Chapter 5). Following recent work on map generation (SLEDGE;[93]), Luke and I were interested in devising an approach for vectorized scene generation using diffusion models, as opposed to rasterizing the vectorized scene. I helped develop the idea through regular discussions with Luke and ran the large-scale training jobs on the compute infrastructure provided by Torc Robotics. Furthermore, I modified the Gpudrive [28] simulator to ensure the PPO policies we trained there can be transferred to work well in the Scenario Dreamer-generated scenarios.

Recent Developments

As the Scenario Dreamer work has only recently been published, it has had a limited impact in the literature. Most of the work that has cited this work is in the realm of long-tail behaviour simulation. At Torc Robotics, this work has been extended to generate temporally consistent scenario rollouts that support sensor simulation for driving [270].

8.3.4 Constrained RL

The work in Chapter 7 was done in collaboration with Julien Roy (First Author), Joshua Romoff, Pierre-Luc Bacon and Christopher Pal. For this work, I contributed to the development of the ideas for behaviour specification, developed the example for naive grid search reward shaping exploration, performed literature review in a joint effort with Julien, and developed the experiments on the OpenWorld environment. Julien Roy was the main architect of the project, conceptualizing the main problem and idea for this project, and the development and experiments on the ArenaEnv. Joshua Romoff, Pierre-Luc Bacon and Chris Pal all provided supervision for this project throughout, as well as helped with the writing.

Fast-forward to 2025, I extend the direct behaviour specification framework to the context of

Vision-Language Models (VLMs) for driving with GRPO. With the collaboration of Rodrigue de Schaetzen, I implemented the behaviour specification framework inside the modified open-source codebase for VLM-R1¹. This modified codebase is a private codebase written by Rodrigue de Schaetzen and Luke Rowe’s Poutine model [264] which employed GRPO on a single reward function to solve the Waymo end-to-end challenge. Instead, I cast the end-to-end autonomous driving problem within the context of direct behaviour specification, and showed its successful application to GRPO for the driving problem, proposing constrained GRPO. As presented in the results of Section 7.4.3, the mathematical analysis I performed provided a principled way of incorporating constraint optimization into the GRPO algorithm. This paper is currently under review.

Context

I had just finished working on AutoBots (Chapter 4) and was ready to focus on a new project. I started talking with Julien about this project in April 2021 while he was interning at Ubisoft. A few months later, I joined him there as an intern as well to focus on this project. Our goal was to provide a method for easily applying RL to video games. In 2025, I am a contributor to a method that performed state-of-the-art on the Waymo End-to-end challenge at the time of the competition, although my role was mainly supervisory. Given that our method applies RL to a problem which has multiple reward components, we pursued its extension to the constrained policy optimization framework I had previously worked on.

Recent Developments

This work has been used by Ubisoft for controlling NPC agents’ behaviours. It has inspired directions in the safe reinforcement learning literature, which deals with the problem of satisfying safety constraints during exploration and deployment, such as avoiding dangerous states as well as limiting risk while still maximizing reward [271].

8.4 Améliorations futures / Future Research

In this section, we propose several directions we hope to explore in the future to expand our knowledge of the field and help push the field of autonomous driving forward.

Constrained Self-Play for Behaviour Simulation Recent work has shown that we can learn realistic agent behaviours through self-play reinforcement learning in pre-recorded initial

¹<https://github.com/om-ai-lab/VLM-R1>

scenes [42, 229]. While Cusumano et al. [229] showed that reward conditioning can give rise to a wide range of agent behaviours during training and this was important for the planner’s final performance, we believe this work can be pushed even further using the behaviour specification framework we proposed in Chapter 7. Concretely, agents can be initialized with varying constraint thresholds, with certain agents having high safety standards while others care more about reaching their goals. Such a framework would enable agent training with heterogeneous agents that tightly respect their behavioural personalities. We believe that this self-play approach unifies the open-ended adversarial training that yields high robustness with the rule-based rigor that produces safe behaviour policies.

Probing Learned Representations in Interactive Driving Policies Modern decision-making policies, from open-loop imitation to closed-loop imitation, self-play reinforcement learning, and constrained RL, are trained under markedly different feedback regimes and seem to produce policies with different characteristics [272]. A future direction is to systematically study how these training regimes shape the *representational content* of the resulting policies, especially under distribution shift, partial observability, and safety constraints. Concretely, one can probe internal policy states for task-relevant variables such as goal intent and, in interactive multi-agent settings, test whether these representations capture predictive structure about *other agents*, e.g., encoding their intents/goals or anticipating their future actions—merely correlating with behavior or instead being *causally used* via targeted interventions and counterfactual rollouts; this was recently done in a hierarchical reinforcement learning setting [273]. This analysis can be extended from privileged vector-state policies to pixel-based, end-to-end planners, where representation quality may determine whether the policy captures the underlying scene dynamics or overfits to spurious visual cues. Finally, mapping scaling laws for representation quality and controllability as a function of model capacity, dataset size, and closed-loop interaction budget could provide a principled lens for choosing between IL, RL, and constrained RL when building reliable driving stacks.

Large-Scale Map Generation We aim to extend the *vectorized latent diffusion model* from Chapter 6. Our current model is limited to an ego-centric field of view of $64\text{m} \times 64\text{m}$. While we can stitch larger areas via inpainting, this introduces three issues: (1) it relies on an ego-agent to define “ahead” and “behind”; (2) it cannot guarantee global consistency under loop closures (e.g., three 90° turns); and (3) agents cannot persist once they exit the original FOV. These limitations constrain long-horizon interaction learning, since the planner cannot reliably encounter and re-encounter agents over extended distances. *Concretely*, addressing this calls for generation that is not tied to a single ego frame, supports loop-consistent topology, and maintains agent continuity beyond the initial window. We view large, consistent, and persistent map generation as a particularly fruitful avenue for further

research.

End-to-End Reasoning and Planning with Task-Relevant Latent World Models We see the next frontier as *reasoning-first autonomy*: agents that maintain beliefs over *task-relevant* latent intentions, roll out counterfactual futures, and plan under explicit uncertainty and constraints. While this thesis largely operated in the abstract-representation regime (with the exception of our constrained GRPO experiments), the path forward is to couple these abstractions with perception via *task-aware latent world models*, i.e., models that learn predictive state abstractions optimized for downstream driving decisions rather than pixel-perfect reconstruction. Concretely, a driving world model should anticipate not only geometric trajectories, but also decision-critical events and affordances (e.g., merging intent, right-of-way resolution, and occlusion-induced uncertainty), enabling planners to reason about neighbouring agents’ goals, the effects of weather and road conditions on dynamics, and the long-horizon consequences of their own actions. Realizing this vision calls for objectives that are *closed-loop and constraint-aware*: the world model should be shaped by the queries a policy needs to act safely and effectively, aligning its latent predictions with downstream control and safety specifications rather than purely reconstructive losses. This naturally motivates agents that actively *query* the world model to evaluate alternative plans through differentiable “what-if” rollouts in latent space, with explicit uncertainty to support risk-sensitive decision-making. Recent directions such as JEPA-style world models [274, 275] provide a promising template for these task-driven objectives by emphasizing prediction in representation space and invariances aligned with decision-relevant structure, offering a principled path toward scalable, reasoning-centric autonomy in driving.

REFERENCES

- [1] A. Obi-obuoha, V. S. Rizama, I. Okafor, H. E. Ovwenkekpere, K. Obe, and J. Ekundayo, “Real-time traffic object detection using detectron 2 with faster r-cnn,” *World Journal of Advanced Research and Reviews*, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:274384910>
- [2] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, “End-to-end object detection with transformers,” *ArXiv*, vol. abs/2005.12872, 2020. [Online]. Available: <https://api.semanticscholar.org/CorpusID:218889832>
- [3] E. Palladin, R. Dietze, P. Narayanan, M. Bijelic, and F. Heide, “Samfusion: Sensor-adaptive multimodal fusion for 3d object detection in adverse weather,” *ArXiv*, vol. abs/2508.16408, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:273234697>
- [4] Y. Li, Y. Chen, X. Qi, Z. Li, J. Sun, and J. Jia, “Unifying voxel-based representation with transformer for 3d object detection,” *ArXiv*, vol. abs/2206.00630, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:249240467>
- [5] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, “nusenes: A multimodal dataset for autonomous driving,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.
- [6] H. Caesar, J. Kabzan, K. S. Tan, W. K. Fong, E. M. Wolff, A. H. Lang, L. Fletcher, O. Beijbom, and S. Omari, “nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles,” *arXiv preprint arXiv.2106.11810*, 2021.
- [7] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research*, vol. 32, pp. 1231 – 1237, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:9455111>
- [8] A. Dosovitskiy, G. Ros, F. Codevilla, A. M. López, and V. Koltun, “Carla: An open urban driving simulator,” in *Conference on Robot Learning*, 2017. [Online]. Available: <https://api.semanticscholar.org/CorpusID:5550767>
- [9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, <http://www.deeplearningbook.org>.

- [10] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, pp. 1735–1780, 1997. [Online]. Available: <https://api.semanticscholar.org/CorpusID:1915014>
- [11] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, “Social lstm: Human trajectory prediction in crowded spaces,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.
- [12] N. Deo and M. Trivedi, “Convolutional social pooling for vehicle trajectory prediction,” *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1549–15498, 2018.
- [13] N. Lee, W. Choi, P. Vernaza, C. Choy, P. Torr, and M. Chandraker, “Desire: Distant future prediction in dynamic scenes with interacting agents,” *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2165–2174, 2017.
- [14] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun, “Implicit latent variable model for scene-consistent motion forecasting,” *arXiv preprint arXiv:2007.12036*, 2020.
- [15] Y. Yuan, X. Weng, Y. Ou, and K. Kitani, “Agentformer: Agent-aware transformers for socio-temporal multi-agent forecasting,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.
- [16] J. Ngiam, B. Caine, V. Vasudevan, Z. Zhang, H.-T. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal *et al.*, “Scene transformer: A unified multi-task model for behavior prediction and planning,” *arXiv preprint arXiv:2106.08417*, 2021.
- [17] H. Zhao, L. Jiang, J. Jia, P. Torr, and V. Koltun, “Point transformer,” 2020.
- [18] R. Girgis, F. Golemo, F. Codevilla, M. Weiss, J. A. D’Souza, S. E. Kahou, F. Heide, and C. J. Pal, “Latent variable sequential set transformers for joint multi-agent motion prediction,” in *International Conference on Learning Representations*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:246824069>
- [19] P. Karkus, B. Ivanovic, S. Mannor, and M. Pavone, “Diffstack: A differentiable and modular control stack for autonomous vehicles,” in *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, ser. Proceedings of Machine Learning Research, vol. 205. PMLR, 2022, pp. 2170–2180.
- [20] Z. Huang, H. Liu, J. Wu, and C. Lv, “Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving,” *IEEE Transactions*

- on Neural Networks and Learning Systems*, vol. 35, pp. 15 222–15 236, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:250920306>
- [21] F. Codevilla, E. Santana, A. M. López, and A. Gaidon, “Exploring the limitations of behavior cloning for autonomous driving,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9329–9338.
 - [22] M. Bansal, A. Krizhevsky, and A. Ogale, “Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst,” *arXiv preprint arXiv:1812.03079*, 2018.
 - [23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, “Playing atari with deep reinforcement learning,” *ArXiv*, vol. abs/1312.5602, 2013. [Online]. Available: <https://api.semanticscholar.org/CorpusID:15238391>
 - [24] D. Dewey, “Reinforcement learning and the reward engineering principle,” in *2014 AAAI Spring Symposium Series*, 2014.
 - [25] J. Roy, R. Girgis, J. Romoff, P.-L. Bacon, and C. J. Pal, “Direct behavior specification via constrained reinforcement learning,” in *International Conference on Machine Learning*, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:245424738>
 - [26] L. Rowe, R. Girgis, A. Gosselin, B. Carrez, F. Golemo, F. Heide, L. Paull, and C. Pal, “CtRL-sim: Reactive and controllable driving agents with offline reinforcement learning,” in *CoRL*, 2024.
 - [27] J. Philion, X. B. Peng, and S. Fidler, “Trajenglish: Learning the language of driving scenarios,” *arXiv preprint arXiv:2312.04535*, 2023.
 - [28] S. Kazemkhani, A. Pandya, D. Cornelisse, B. Shacklett, and E. Vinitsky, “Gpudrive: Data-driven, multi-agent driving simulation at 1 million fps,” *ArXiv*, vol. abs/2408.01584, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:271710269>
 - [29] N. Montali, J. Lambert, P. Mougin, A. Kuefler, N. Rhinehart, M. Li, C. Gulino, T. Emrich, Z. Yang, S. Whiteson, B. A. White, and D. Anguelov, “The waymo open sim agents challenge,” *ArXiv*, vol. abs/2305.12032, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:258832394>

- [30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [31] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer; 1st ed. 2006, 2006.
- [32] M. Treiber, A. Hennecke, and D. Helbing, “Congested traffic states in empirical observations and microscopic simulations,” *Physical review E*, vol. 62, no. 2, p. 1805, 2000.
- [33] A. Kesting, M. Treiber, and D. Helbing, “General lane-changing model mobil for car-following models,” *Transportation Research Record*, vol. 1999, no. 1, pp. 86–94, 2007.
- [34] S. Suo, S. Regalado, S. Casas, and R. Urtasun, “Trafficsim: Learning to simulate realistic multi-agent behaviors,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 400–10 409.
- [35] D. Xu, Y. Chen, B. Ivanovic, and M. Pavone, “BITS: bi-level imitation for traffic simulation,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2023.
- [36] Z. Zhong, D. Rempe, D. Xu, Y. Chen, S. Veer, T. Che, B. Ray, and M. Pavone, “Guided conditional diffusion for controllable traffic simulation,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2023.
- [37] W. Wu, X. Feng, Z. Gao, and Y. Kan, “Smart: Scalable multi-agent real-time motion generation via next-token prediction,” *Advances in Neural Information Processing Systems*, vol. 37, pp. 114 048–114 071, 2024.
- [38] E. Vinitzky, N. Lichtlé, X. Yang, B. Amos, and J. Foerster, “Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [39] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. Zhou *et al.*, “Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9710–9719.
- [40] K. Lee, O. Nachum, M. Yang, L. Lee, D. Freeman, S. Guadarrama, I. Fischer, W. Xu, E. Jang, H. Michalewski, and I. Mordatch, “Multi-game decision transformers,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.

- [41] A. Piché, R. Pardinás, D. Vazquez, and C. Pal, “A probabilistic perspective on reinforcement learning via supervised learning,” in *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.
- [42] D. Cornelisse, A. Pandya, K. Joseph, J. Su’arez, and E. Vinitzky, “Building reliable sim driving agents by scaling self-play,” *ArXiv*, vol. abs/2502.14706, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:276482379>
- [43] C. Gulino, J. Fu, W. Luo, G. Tucker, E. Bronstein, Y. Lu, J. Harb, X. Pan, Y. Wang, X. Chen *et al.*, “Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 7730–7742, 2023.
- [44] L. Rowe, R. Girgis, A. Gosselin, L. Paull, C. Pal, and F. Heide, “Scenario dreamer: Vectorized latent diffusion for generating driving simulation environments,” *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17 207–17 218, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:277435420>
- [45] K. Cho, B. V. Merriënboer, Çağlar Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder–decoder for statistical machine translation,” *ArXiv*, vol. abs/1406.1078, 2014.
- [46] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” *arXiv preprint arXiv:1409.3215*, 2014.
- [47] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *CoRR*, vol. abs/1409.0473, 2014. [Online]. Available: <https://api.semanticscholar.org/CorpusID:11212020>
- [48] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NIPS*, 2017.
- [49] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, “Set transformer: A framework for attention-based permutation-invariant neural networks,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 3744–3753.
- [50] J. L. Ba, J. R. Kiros, and G. E. Hinton, “Layer normalization,” *arXiv preprint arXiv:1607.06450*, 2016.

- [51] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe, “Variational inference: A review for statisticians,” *Journal of the American statistical Association*, vol. 112, no. 518, pp. 859–877, 2017.
- [52] S. Gershman and N. Goodman, “Amortized inference in probabilistic reasoning,” in *Proceedings of the annual meeting of the cognitive science society*, vol. 36, no. 36, 2014.
- [53] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” *arXiv preprint arXiv:1312.6114*, 2013.
- [54] J. N. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep unsupervised learning using nonequilibrium thermodynamics,” *ArXiv*, vol. abs/1503.03585, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:14888175>
- [55] Y. Song and S. Ermon, “Generative modeling by estimating gradients of the data distribution,” in *Neural Information Processing Systems*, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:196470871>
- [56] J. Ho, A. Jain, and P. Abbeel, “Denoising diffusion probabilistic models,” in *NeurIPS*, 2020.
- [57] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, “High-resolution image synthesis with latent diffusion models,” in *CVPR*, 2022.
- [58] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
- [59] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, “Soft actor-critic algorithms and applications,” *arXiv preprint arXiv:1812.05905*, 2018.
- [60] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [61] S. Fujimoto, H. Hoof, and D. Meger, “Addressing function approximation error in actor-critic methods,” in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.
- [62] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.

- [63] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, “Trust region policy optimization,” *ArXiv*, vol. abs/1502.05477, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:16046818>
- [64] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” *CoRR*, vol. abs/1506.02438, 2015. [Online]. Available: <https://api.semanticscholar.org/CorpusID:3075448>
- [65] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe, “Training language models to follow instructions with human feedback,” 2022. [Online]. Available: <https://arxiv.org/abs/2203.02155>
- [66] Z. Shao, P. Wang, Q. Zhu, R. Xu, J.-M. Song, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, “Deepseekmath: Pushing the limits of mathematical reasoning in open language models,” *ArXiv*, vol. abs/2402.03300, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:267412607>
- [67] Y. Li, M. Tian, D. Zhu, J. Zhu, Z. Lin, Z. Xiong, and X. Zhao, “Drive-rl: Bridging reasoning and planning in vlms for autonomous driving with reinforcement learning,” *ArXiv*, vol. abs/2506.18234, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:279999326>
- [68] K. Chen, W. Sun, H. Cheng, and S. Zheng, “Rift: Group-relative rl fine-tuning for realistic and controllable traffic simulation,” 2025. [Online]. Available: <https://arxiv.org/abs/2505.03344>
- [69] M. Pei, S. Shi, and S. Shen, “Advancing multi-agent traffic simulation via rl-style reinforcement fine-tuning,” *ArXiv*, vol. abs/2509.23993, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:281675127>
- [70] E. Leurent, “An environment for autonomous driving decision-making,” <https://github.com/eleurent/highway-env>, 2018.
- [71] P. Cai, Y. Lee, Y. Luo, and D. Hsu, “SUMMIT: A simulator for urban driving in massive mixed traffic,” in *ICRA*, 2020.
- [72] D. Chen, M. Zhu, H. Yang, X. Wang, and Y. Wang, “Data-driven traffic simulation: A comprehensive review,” *IEEE Trans. Intell. Veh.*, 2024.

- [73] Z. Yang, Y. Chen, J. Wang, S. Manivasagam, W. Ma, A. J. Yang, and R. Urtasun, “Unisim: A neural closed-loop sensor simulator,” in *CVPR*, 2023.
- [74] J. Yang, B. Ivanovic, O. Litany, X. Weng, S. W. Kim, B. Li, T. Che, D. Xu, S. Fidler, M. Pavone, and Y. Wang, “Emernerf: Emergent spatial-temporal scene decomposition via self-supervision,” *ICLR*, 2024.
- [75] Y. Yan, H. Lin, C. Zhou, W. Wang, H. Sun, K. Zhan, X. Lang, X. Zhou, and S. Peng, “Street gaussians for modeling dynamic urban scenes,” in *ECCV*, 2024.
- [76] W. Ljungbergh, A. Tonderski, J. Johnander, H. Caesar, K. Åström, M. Felsberg, and C. Petersson, “Neuroncap: Photorealistic closed-loop safety testing for autonomous driving,” *ECCV*, 2024.
- [77] X. Zhou, Z. Lin, X. Shan, Y. Wang, D. Sun, and M. Yang, “Drivinggaussian: Composite gaussian splatting for surrounding dynamic autonomous driving scenes,” in *CVPR*, 2024.
- [78] H. Zhou, J. Shao, L. Xu, D. Bai, W. Qiu, B. Liu, Y. Wang, A. Geiger, and Y. Liao, “HUGS: holistic urban 3d scene understanding via gaussian splatting,” in *CVPR*, 2024.
- [79] H. Zhou, L. Lin, J. Wang, Y. Lu, D. Bai, B. Liu, Y. Wang, A. Geiger, and Y. Liao, “HUGSIM: A real-time, photo-realistic and closed-loop simulator for autonomous driving,” *arXiv*, 2024.
- [80] Z. Xie, J. Zhang, W. Li, F. Zhang, and L. Zhang, “S-nerf: Neural radiance fields for street views,” in *ICLR*, 2023.
- [81] Y. Chen, J. Zhang, Z. Xie, W. Li, F. Zhang, J. Lu, and L. Zhang, “S-nerf++: Autonomous driving simulation via neural reconstruction and generation,” *IEEE TPAMI*, 2025.
- [82] A. Tonderski, C. Lindström, G. Hess, W. Ljungbergh, L. Svensson, and C. Petersson, “Neurad: Neural rendering for autonomous driving,” in *CVPR*, 2024.
- [83] J. Ost, F. Mannan, N. Thuerey, J. Knodt, and F. Heide, “Neural scene graphs for dynamic scenes,” in *CVPR*, 2021.
- [84] Q. Li, Z. Peng, L. Feng, Q. Zhang, Z. Xue, and B. Zhou, “Metadrive: Composing diverse driving scenarios for generalizable reinforcement learning,” *IEEE TPAMI*, 2023.

- [85] J. W. Lavington, K. Zhang, V. Lioutas, M. Niedoba, Y. Liu, D. Green, S. Naderiparizi, X. Liang, S. Dabiri, A. Ścibior, B. Zwartsenberg, and F. Wood, “Torchdriveenv: A reinforcement learning benchmark for autonomous driving with reactive, realistic, and diverse non-playable characters,” *arXiv*, 2024.
- [86] N. Karnchanachari, D. Geromichalos, K. S. Tan, N. Li, C. Eriksen, S. Yaghoubi, N. Mehdipour, G. Bernasconi, W. K. Fong, Y. Guo, and H. Caesar, “Towards learning-based planning: The nuplan benchmark for real-world autonomous driving,” in *ICRA*, 2024.
- [87] S. Kazemkhani, A. Pandya, D. Cornelisse, B. Shacklett, and E. Vinitzky, “GPUDrive: Data-driven, multi-agent driving simulation at 1 million FPS,” in *ICLR*, 2025.
- [88] M. Zhou, J. Luo, J. Villella, Y. Yang, D. Rusu, J. Miao, W. Zhang, M. Alban, I. Fadakar, Z. Chen, A. C. Huang, Y. Wen, K. Hassanzadeh, D. Graves, D. Chen, Z. Zhu, N. Nguyen, M. Elsayed, K. Shao, S. Ahilan, B. Zhang, J. Wu, Z. Fu, K. Rezaee, P. Yadmellat, M. Rohani, N. P. Nieves, Y. Ni, S. Banijamali, A. C. Rivers, Z. Tian, D. Palenicek, H. bou Ammar, H. Zhang, W. Liu, J. Hao, and J. Wang, “Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving,” 2020.
- [89] A. Hu, L. Russell, H. Yeo, Z. Murez, G. Fedoseev, A. Kendall, J. Shotton, and G. Corrado, “GAIA-1: A generative world model for autonomous driving,” *arXiv*, 2023.
- [90] X. Yang, L. Wen, Y. Ma, J. Mei, X. Li, T. Wei, W. Lei, D. Fu, P. Cai, M. Dou, B. Shi, L. He, Y. Liu, and Y. Qiao, “Drivearena: A closed-loop generative simulation platform for autonomous driving,” *arXiv*, 2024.
- [91] S. Gao, J. Yang, L. Chen, K. Chitta, Y. Qiu, A. Geiger, J. Zhang, and H. Li, “Vista: A generalizable driving world model with high fidelity and versatile controllability,” in *NeurIPS*, 2024.
- [92] S. Sun, Z. Gu, T. Sun, J. Sun, C. Yuan, Y. Han, D. Li, and M. H. Ang, “Drivescenegen: Generating diverse and realistic driving scenarios from scratch,” *IEEE RA-L*, 2024.
- [93] K. Chitta, D. Dauner, and A. Geiger, “Sledge: Synthesizing driving environments with generative models and rule-based traffic,” in *European Conference on Computer Vision*, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:268691934>
- [94] J. Bruce, M. D. Dennis, A. Edwards, J. Parker-Holder, Y. Shi, E. Hughes, M. Lai, A. Mavalankar, R. Steigerwald, C. Apps, Y. Aytar, S. Bechtle, F. M. P. Behbahani,

- S. C. Y. Chan, N. Heess, L. Gonzalez, S. Osindero, S. Ozair, S. E. Reed, J. Zhang, K. Zolna, J. Clune, N. de Freitas, S. Singh, and T. Rocktäschel, “Genie: Generative interactive environments,” in *ICML*, 2024.
- [95] E. Alonso, A. Jelley, V. Micheli, A. Kanervisto, A. J. Storkey, T. Pearce, and F. Fleuret, “Diffusion for world modeling: Visual details matter in atari,” *NeurIPS*, 2024.
- [96] D. Valevski, Y. Leviathan, M. Arar, and S. Fruchter, “Diffusion models are real-time game engines,” in *ICLR*, 2025.
- [97] S. Yang, Y. Du, S. K. S. Ghasemipour, J. Tompson, L. P. Kaelbling, D. Schuurmans, and P. Abbeel, “Learning interactive real-world simulators,” in *ICLR*, 2024.
- [98] Y. Yu, J. Xu, and L. Liu, “Adaptive diffusion terrain generator for autonomous uneven terrain navigation,” in *CoRL*, 2024.
- [99] J. Wu, S. Yin, N. Feng, X. He, D. Li, J. Hao, and M. Long, “ivideogpt: Interactive videogpts are scalable world models,” in *NeurIPS*, 2024.
- [100] S. Tan, K. Wong, S. Wang, S. Manivasagam, M. Ren, and R. Urtasun, “Scenegen: Learning to generate realistic traffic scenes,” in *CVPR*, 2021.
- [101] J. Lu, K. Wong, C. Zhang, S. Suo, and R. Urtasun, “Scenecontrol: Diffusion for controllable traffic scene generation,” in *ICRA*, 2024.
- [102] L. Mi, H. Zhao, C. Nash, X. Jin, J. Gao, C. Sun, C. Schmid, N. Shavit, Y. Chai, and D. Anguelov, “Hdmapgen: A hierarchical graph generative model of high definition maps,” in *CVPR*, 2021.
- [103] J. Gao, C. Sun, H. Zhao, Y. Shen, D. Anguelov, C. Li, and C. Schmid, “Vectornet: Encoding hd maps and agent dynamics from vectorized representation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 11 525–11 533.
- [104] E. Hoogeboom, V. G. Satorras, C. Vignac, and M. Welling, “Equivariant diffusion for molecule generation in 3d,” in *ICML*, 2022.
- [105] M. Xu, A. S. Powers, R. O. Dror, S. Ermon, and J. Leskovec, “Geometric latent diffusion models for 3d molecule generation,” in *ICML*, 2023.
- [106] J. Chen, R. Deng, and Y. Furukawa, “Polydiffuse: Polygonal shape reconstruction via guided set diffusion models,” in *NeurIPS*, 2023.

- [107] B. Jiang, S. Chen, Q. Xu, B. Liao, J. Chen, H. Zhou, Q. Zhang, W. Liu, C. Huang, and X. Wang, “VAD: vectorized scene representation for efficient autonomous driving,” in *ICCV*, 2023.
- [108] S. Chen, B. Jiang, H. Gao, B. Liao, Q. Xu, Q. Zhang, C. Huang, W. Liu, and X. Wang, “Vadv2: End-to-end vectorized autonomous driving via probabilistic planning,” *arXiv*, 2024.
- [109] J. Ngiam, V. Vasudevan, B. Caine, Z. Zhang, H. L. Chiang, J. Ling, R. Roelofs, A. Bewley, C. Liu, A. Venugopal, D. J. Weiss, B. Sapp, Z. Chen, and J. Shlens, “Scene transformer: A unified architecture for predicting future trajectories of multiple agents,” in *International Conference on Learning Representations (ICLR)*, 2022.
- [110] R. Girgis, F. Golemo, F. Codevilla, M. Weiss, J. A. D’Souza, S. E. Kahou, F. Heide, and C. J. Pal, “Latent variable sequential set transformers for joint multi-agent motion prediction,” in *ICLR*, 2022.
- [111] L. Rowe, M. Ethier, E.-H. Dykhne, and K. Czarnecki, “FJMP: factorized joint multi-agent motion prediction over learned directed acyclic interaction graphs,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [112] Z. Zhou, L. Ye, J. Wang, K. Wu, and K. Lu, “Hivt: Hierarchical vector transformer for multi-agent motion prediction,” in *CVPR*, 2022.
- [113] B. Liao, S. Chen, X. Wang, T. Cheng, Q. Zhang, W. Liu, and C. Huang, “Maptr: Structured modeling and learning for online vectorized HD map construction,” in *ICLR*, 2023.
- [114] B. Liao, S. Chen, Y. Zhang, B. Jiang, Q. Zhang, W. Liu, C. Huang, and X. Wang, “Maptrv2: An end-to-end framework for online vectorized HD map construction,” *Int. J. Comput. Vis.*, 2025.
- [115] C. M. Jiang, A. Cornman, C. Park, B. Sapp, Y. Zhou, and D. Anguelov, “Motiondiffuser: Controllable multi-agent motion prediction using diffusion,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [116] L. Bergamini, Y. Ye, O. Scheel, L. Chen, C. Hu, L. D. Pero, B. Osinski, H. Grimmer, and P. Ondruska, “Simnet: Learning reactive self-driving simulations from real-world observations,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2021.

- [117] A. Kamenev, L. Wang, O. B. Bohan, I. Kulkarni, B. Kartal, A. Molchanov, S. Birchfield, D. Nistér, and N. Smolyanskiy, “Predictionnet: Real-time joint probabilistic traffic prediction for planning, control, and simulation,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2022.
- [118] A. Ścibior, V. Lioutas, D. Reda, P. Bateni, and F. Wood, “Imagining the road ahead: Multi-agent trajectory prediction via differentiable simulation,” in *Proceedings of the IEEE International Intelligent Transportation Systems Conference (ITSC)*, 2021.
- [119] Y. Wang, T. Zhao, and F. Yi, “Multiverse transformer: 1st place solution for waymo open sim agents challenge 2023,” *arXiv preprint arXiv.2306.11868*, 2023.
- [120] S. Casas, C. Gulino, S. Suo, K. Luo, R. Liao, and R. Urtasun, “Implicit latent variable model for scene-consistent motion forecasting,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2020.
- [121] Y. Chen, B. Ivanovic, and M. Pavone, “Scept: Scene-consistent, policy-based trajectory predictions for planning,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.
- [122] J. Ho and S. Ermon, “Generative adversarial imitation learning,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2016.
- [123] R. P. Bhattacharyya, D. J. Phillips, B. Wulfe, J. Morton, A. Kuefler, and M. J. Kochenderfer, “Multi-agent imitation learning for driving simulation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.
- [124] G. Zheng, H. Liu, K. Xu, and Z. Li, “Objective-aware traffic simulation via inverse reinforcement learning,” in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2021.
- [125] M. Igl, D. Kim, A. Kuefler, P. Mougin, P. Shah, K. Shiarlis, D. Anguelov, M. Palatucci, B. White, and S. Whiteson, “Symphony: Learning realistic and diverse agents for autonomous driving simulation,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2022.
- [126] Y. Lu, J. Fu, G. Tucker, X. Pan, E. Bronstein, R. Roelofs, B. Sapp, B. White, A. Faust, S. Whiteson, D. Anguelov, and S. Levine, “Imitation is not enough: Robustifying imitation with reinforcement learning for challenging driving scenarios,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.

- [127] C. Zhang, J. Tu, L. Zhang, K. Wong, S. Suo, and R. Urtasun, “Learning realistic traffic agents in closed-loop,” in *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, ser. Proceedings of Machine Learning Research, vol. 229. PMLR, 2023, pp. 800–821.
- [128] D. Rempe, J. Philion, L. J. Guibas, S. Fidler, and O. 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 (CVPR)*, 2022.
- [129] Z. Zhang, A. Liniger, D. Dai, F. Yu, and L. V. Gool, “Trafficbots: Towards world models for autonomous driving simulation and motion prediction,” in *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2023.
- [130] S. Suo, K. Wong, J. Xu, J. Tu, A. Cui, S. Casas, and R. Urtasun, “MIXSIM: A hierarchical framework for mixed reality traffic simulation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023.
- [131] W. Chang, C. Tang, C. Li, Y. Hu, M. Tomizuka, and W. Zhan, “Editing driver character: Socially-controllable behavior generation for interactive traffic simulation,” *IEEE Robotics Autom. Lett.*, 2023.
- [132] Z. Zhong, D. Rempe, Y. Chen, B. Ivanovic, Y. Cao, D. Xu, M. Pavone, and B. Ray, “Language-guided traffic simulation via scene-level diffusion,” *arXiv preprint arXiv.2306.06344*, 2023.
- [133] W. Chang, F. Pittaluga, M. Tomizuka, W. Zhan, and M. Chandraker, “Controllable safety-critical closed-loop traffic simulation via guided diffusion,” *arXiv preprint arXiv.2401.00391*, 2024.
- [134] C. Xu, D. Zhao, A. Sangiovanni-Vincentelli, and B. Li, “Diffscene: Diffusion-based safety-critical scenario generation for autonomous vehicles,” in *The Second Workshop on New Frontiers in Adversarial Machine Learning*, 2023.
- [135] Z. Guo, X. Gao, J. Zhou, X. Cai, and B. Shi, “Scenedm: Scene-level multi-agent trajectory generation with consistent diffusion models,” *arXiv preprint: arXiv.2311.15736*, 2023.
- [136] W. Ding, Y. Cao, D. Zhao, C. Xiao, and M. Pavone, “Realgen: Retrieval augmented generation for controllable traffic scenarios,” *arXiv preprint arXiv.2312.13303*, 2023.

- [137] Y. AbeySirigoonawardena, F. Shkurti, and G. Dudek, “Generating adversarial driving scenarios in high-fidelity simulators,” in *International Conference on Robotics and Automation (ICRA)*, 2019.
- [138] J. Wang, A. Pun, J. Tu, S. Manivasagam, A. Sadat, S. Casas, M. Ren, and R. Urtasun, “Advsim: Generating safety-critical scenarios for self-driving vehicles,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.
- [139] N. Gontier, P. R. López, I. H. Laradji, D. Vázquez, and C. J. Pal, “Language decision transformers with exponential tilt for interactive text environments,” *arXiv preprint arXiv:2302.05507*, 2023.
- [140] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, “Decision transformer: Reinforcement learning via sequence modeling,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.
- [141] R. K. Srivastava, P. Shyam, F. W. Mutz, W. Jaśkowski, and J. Schmidhuber, “Training agents using upside-down reinforcement learning,” *ArXiv*, vol. abs/1912.02877, 2019. [Online]. Available: <https://api.semanticscholar.org/CorpusID:208857468>
- [142] A. Kumar, X. B. Peng, and S. Levine, “Reward-conditioned policies,” *arXiv preprint arXiv:1912.13465*, 2019.
- [143] S. Emmons, B. Eysenbach, I. Kostrikov, and S. Levine, “Rvs: What is essential for offline rl via supervised learning?” *ArXiv*, vol. abs/2112.10751, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:245334837>
- [144] L. Meng, M. Wen, Y. Yang, C. Le, X. Li, W. Zhang, Y. Wen, H. Zhang, J. Wang, and B. Xu, “Offline pre-trained multi-agent decision transformer: One big sequence model tackles all smac tasks,” *ArXiv*, vol. abs/2112.02845, 2021. [Online]. Available: <https://api.semanticscholar.org/CorpusID:245335360>
- [145] E. Altman, *Constrained Markov decision processes*. CRC Press, 1999, vol. 7.
- [146] V. S. Borkar, “An actor-critic algorithm for constrained markov decision processes,” *Systems & control letters*, vol. 54, no. 3, pp. 207–213, 2005.
- [147] C. Tessler, D. J. Mankowitz, and S. Mannor, “Reward constrained policy optimization,” *arXiv preprint arXiv:1805.11074*, 2018.

- [148] A. Stooke, J. Achiam, and P. Abbeel, “Responsive safety in reinforcement learning by pid lagrangian methods,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 9133–9143.
- [149] J. Achiam, D. Held, A. Tamar, and P. Abbeel, “Constrained policy optimization,” in *International Conference on Machine Learning*. PMLR, 2017, pp. 22–31.
- [150] Y. Chow, O. Nachum, A. Faust, E. Duenez-Guzman, and M. Ghavamzadeh, “Lyapunov-based safe policy optimization for continuous control,” *arXiv preprint arXiv:1901.10031*, 2019.
- [151] T.-Y. Yang, J. Rosca, K. Narasimhan, and P. J. Ramadge, “Projection-based constrained policy optimization,” *arXiv preprint arXiv:2010.03152*, 2020.
- [152] Y. Zhang, Q. Vuong, and K. W. Ross, “First order constrained optimization in policy space,” *arXiv preprint arXiv:2002.06506*, 2020.
- [153] G. Dalal, K. Dvijotham, M. Vecerik, T. Hester, C. Paduraru, and Y. Tassa, “Safe exploration in continuous action spaces,” *arXiv preprint arXiv:1801.08757*, 2018.
- [154] D. A. Calian, D. J. Mankowitz, T. Zahavy, Z. Xu, J. Oh, N. Levine, and T. Mann, “Balancing constraints and rewards with meta-gradient d4pg,” *arXiv preprint arXiv:2010.06324*, 2020.
- [155] A. Ray, J. Achiam, and D. Amodei, “Benchmarking safe exploration in deep reinforcement learning,” *arXiv preprint arXiv:1910.01708*, vol. 7, 2019.
- [156] B. Zheng, S. Verma, J. Zhou, I. Tsang, and F. Chen, “Imitation learning: Progress, taxonomies and opportunities,” *arXiv preprint arXiv:2106.12177*, 2021.
- [157] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, “Deep reinforcement learning from human preferences,” *arXiv preprint arXiv:1706.03741*, 2017.
- [158] W. Saunders, G. Sastry, A. Stuhlmueeller, and O. Evans, “Trial without error: Towards safe reinforcement learning via human intervention,” *arXiv preprint arXiv:1707.05173*, 2017.
- [159] P. Goyal, S. Niekum, and R. J. Mooney, “Using natural language for reward shaping in reinforcement learning,” *arXiv preprint arXiv:1903.02020*, 2019.
- [160] J. MacGlashan, M. Babes-Vroman, M. desJardins, M. L. Littman, S. Muresan, S. Squire, S. Tellex, D. Arumugam, and L. Yang, “Grounding english commands to reward functions.” in *Robotics: Science and Systems*, 2015.

- [161] D. Hadfield-Menell, S. Milli, P. Abbeel, S. Russell, and A. Dragan, “Inverse reward design,” *arXiv preprint arXiv:1711.02827*, 2017.
- [162] S. Mindermann, R. Shah, A. Gleave, and D. Hadfield-Menell, “Active inverse reward design,” *arXiv preprint arXiv:1809.03060*, 2018.
- [163] E. Ratner, D. Hadfield-Menell, and A. D. Dragan, “Simplifying reward design through divide-and-conquer,” *arXiv preprint arXiv:1806.02501*, 2018.
- [164] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, “A survey of deep reinforcement learning in video games,” *arXiv preprint arXiv:1912.10944*, 2019.
- [165] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, “Dota 2 with large scale deep reinforcement learning,” *arXiv preprint arXiv:1912.06680*, 2019.
- [166] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, “Grandmaster level in starcraft ii using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.
- [167] M. Jacob, S. Devlin, and K. Hofmann, ““it’s unwieldy and it takes a lot of time”—challenges and opportunities for creating agents in commercial games,” in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, vol. 16, 2020, pp. 88–94.
- [168] E. Alonso, M. Peter, D. Goumard, and J. Romoff, “Deep reinforcement learning for navigation in aaa video games,” *arXiv preprint arXiv:2011.04764*, 2020.
- [169] S. Devlin, R. Georgescu, I. Momennejad, J. Rzepecki, E. Zuniga, G. Costello, G. Leroy, A. Shaw, and K. Hofmann, “Navigation turing test (ntt): Learning to evaluate human-like navigation,” *arXiv preprint arXiv:2105.09637*, 2021.
- [170] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslén, “Augmenting automated game testing with deep reinforcement learning,” in *2020 IEEE Conference on Games (CoG)*. IEEE, 2020, pp. 600–603.
- [171] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, “Improving playtesting coverage via curiosity driven reinforcement learning agents,” *arXiv preprint arXiv:2103.13798*, 2021.

- [172] P. L. P. de Woillemont, R. Labory, and V. Corruble, “Configurable agent with reward as input: A play-style continuum generation,” in *2021 IEEE Conference on Games (CoG)*. IEEE, 2021, pp. 1–8.
- [173] L. Gisslén, A. Eakins, C. Gordillo, J. Bergdahl, and K. Tollmar, “Adversarial reinforcement learning for procedural content generation,” *arXiv preprint arXiv:2103.04847*, 2021.
- [174] T. Moskovitz, A. K. Singh, D. Strouse, T. Sandholm, R. Salakhutdinov, A. D. Dragan, and S. M. McAleer, “Confronting reward model overoptimization with constrained rlhf,” *ArXiv*, vol. abs/2310.04373, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:263829192>
- [175] B. Zhang, Y. Zhang, J. Ji, Y. Lei, J. Dai, Y. Chen, and Y. Yang, “Safevla: Towards safety alignment of vision-language-action model via safe reinforcement learning,” *ArXiv*, vol. abs/2503.03480, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:276782535>
- [176] J. Dai, X. Pan, R. Sun, J. Ji, X. Xu, M. Liu, Y. Wang, and Y. Yang, “Safe rlhf: Safe reinforcement learning from human feedback,” *ArXiv*, vol. abs/2310.12773, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:264306078>
- [177] J. Ji, X. Chen, R. Pan, H. Zhu, C. Zhang, J. Li, D. Hong, B. Chen, J. Zhou, K. Wang, J. Dai, C.-M. Chan, S. Han, Y. Guo, and Y. Yang, “Safe rlhf-v: Safe reinforcement learning from human feedback in multimodal large language models,” *ArXiv*, vol. abs/2503.17682, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:277271998>
- [178] Y. Bai, S. Kadavath, S. Kundu, A. Askeel, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, C. Chen, C. Olsson, C. Olah, D. Hernandez, D. Drain, D. Ganguli, D. Li, E. Tran-Johnson, E. Perez, J. Kerr, J. Mueller, J. Ladish, J. Landau, K. Ndousse, K. Lukosuite, L. Lovitt, M. Sellitto, N. Elhage, N. Schiefer, N. Mercado, N. Dassarma, R. Lasenby, R. Larson, S. Ringer, S. Johnston, S. Kravec, S. E. Showk, S. Fort, T. Lanham, T. Telleen-Lawton, T. Conerly, T. J. Henighan, T. Hume, S. Bowman, Z. Hatfield-Dodds, B. Mann, D. Amodei, N. Joseph, S. McCandlish, T. B. Brown, and J. Kaplan, “Constitutional ai: Harmlessness from ai feedback,” *ArXiv*, vol. abs/2212.08073, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:254823489>

- [179] H. Wang, W. Xiong, T. Xie, H. Zhao, and T. Zhang, “Interpretable preferences via multi-objective reward modeling and mixture-of-experts,” in *Conference on Empirical Methods in Natural Language Processing*, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:270562658>
- [180] Z. Liu, C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin, “Understanding r1-zero-like training: A critical perspective,” *arXiv preprint arXiv:2503.20783*, 2025.
- [181] G. Li, M. Lin, T. Galanti, Z. Tu, and T. Yang, “Disco: Reinforcing large reasoning models with discriminative constrained optimization,” *arXiv preprint arXiv:2505.12366*, 2025.
- [182] X. Li, Z. Li, Y. Kosuga, and V. Bian, “Optimizing safe and aligned language generation: A multi-objective grpo approach,” *ArXiv*, vol. abs/2503.21819, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:277435738>
- [183] Y. Ichihara, Y. Jinnai, T. Morimura, M. Sakamoto, R. Mitsuhashi, and E. Uchibe, “Mo-grpo: Mitigating reward hacking of group relative policy optimization on multi-objective problems,” *arXiv preprint arXiv:2509.22047*, 2025.
- [184] X. Tang, M. Kan, S. Shan, and X. Chen, “Plan-r1: Safe and feasible trajectory planning as language modeling,” *ArXiv*, vol. abs/2505.17659, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:278887056>
- [185] S.-Y. Liu, X. Dong, X. Lu, S. Diao, P. Belcak, M. Liu, M.-H. Chen, H. Yin, Y.-C. F. Wang, K.-T. Cheng *et al.*, “Gdpo: Group reward-decoupled normalization policy optimization for multi-reward rl optimization,” *arXiv preprint arXiv:2601.05242*, 2026.
- [186] A. Kembhavi, M. Seo, D. Schwenk, J. Choi, A. Farhadi, and H. Hajishirzi, “Are you smarter than a sixth grader? textbook question answering for multimodal machine comprehension,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5376–5384.
- [187] M. Richardson, C. J. Burges, and E. Renshaw, “MCTest: A challenge dataset for the open-domain machine comprehension of text,” in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*. Seattle, Washington, USA: Association for Computational Linguistics, Oct. 2013, pp. 193–203. [Online]. Available: <https://aclanthology.org/D13-1020>

- [188] Y. Biktairov, M. Stebelev, I. Rudenko, O. Shliazhko, and B. Yangel, “PRANK: motion prediction based on ranking,” *CoRR*, vol. abs/2010.12007, 2020. [Online]. Available: <https://arxiv.org/abs/2010.12007>
- [189] A. Sadeghian, V. Kosaraju, A. Gupta, S. Savarese, and A. Alahi, “Trajnet: Towards a benchmark for human trajectory prediction,” *arXiv preprint*, 2018.
- [190] H. Caesar, V. Bankiti, A. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, “nusenes: A multimodal dataset for autonomous driving,” *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 618–11 628, 2020.
- [191] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, “Argoverse: 3d tracking and forecasting with rich maps,” *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8740–8749, 2019.
- [192] P. Kothari, S. Kreiss, and A. Alahi, “Human trajectory forecasting in crowds: A deep learning perspective,” *IEEE Transactions on Intelligent Transportation Systems*, 2021.
- [193] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, “Human-level concept learning through probabilistic program induction,” *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.
- [194] C. Tang and R. R. Salakhutdinov, “Multiple futures prediction,” in *Advances in Neural Information Processing Systems*, 2019, pp. 15 424–15 434.
- [195] K. Messaoud, N. Deo, M. M. Trivedi, and F. Nashashibi, “Multi-head attention with joint agent-map representation for trajectory prediction in autonomous driving,” *arXiv*, pp. arXiv–2005, 2020.
- [196] T. Salzmann, B. Ivanovic, P. Chakravarty, and M. Pavone, “Trajectron++: Multi-agent generative trajectory forecasting with heterogeneous data for control,” *ArXiv*, vol. abs/2001.03093, 2020.
- [197] T. Phan-Minh, E. Grigore, F. Boulton, O. Beijbom, and E. M. Wolff, “Covernet: Multimodal behavior prediction using trajectory sets,” *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14 062–14 071, 2020.
- [198] S. Khandelwal, W. Qi, J. Singh, A. Hartnett, and D. Ramanan, “What-if motion prediction for autonomous driving,” 2020.

- [199] T. Gilles, S. Sabatini, D. Tsishkou, B. Stanciulescu, and F. Moutarde, “Go-home: Graph-oriented heatmap output for future motion estimation,” *arXiv preprint arXiv:2109.01827*, 2021.
- [200] J. Mercat, T. Gilles, N. Zoghby, G. Sandou, D. Beauvois, and G. Gil, “Multi-head attention for joint multi-modal vehicle motion forecasting,” in *IEEE International Conference on Robotics and Automation*, 2020.
- [201] H. Zhao, J. Gao, T. Lan, C. Sun, B. Sapp, B. Varadarajan, Y. Shen, Y. Chai, C. Schmid, C. Li, and D. Anguelov, “Tnt: Target-driven trajectory prediction,” *ArXiv*, vol. abs/2008.08294, 2020.
- [202] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, “Learning lane graph representations for motion forecasting,” *ArXiv*, vol. abs/2007.13732, 2020.
- [203] M. Ye, T. Cao, and Q. Chen, “Tpcn: Temporal point cloud networks for motion forecasting,” *ArXiv*, vol. abs/2103.03067, 2021.
- [204] Y. Liu, J. Zhang, L. Fang, Q. Jiang, and B. Zhou, “Multimodal motion prediction with stacked transformers,” *ArXiv*, vol. abs/2103.11624, 2021.
- [205] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Póczos, R. Salakhutdinov, and A. Smola, “Deep sets,” *arXiv preprint arXiv:1703.06114*, 2017.
- [206] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.
- [207] J. S. Hartford, J. R. Wright, and K. Leyton-Brown, “Deep learning for predicting human strategic behavior,” in *NIPS*, 2016.
- [208] L. L. Li, B. Yang, M. Liang, W. Zeng, M. Ren, S. Segal, and R. Urtasun, “End-to-end contextual perception and prediction with interaction transformer,” *arXiv preprint arXiv:2008.05927*, 2020.
- [209] J. Li, F. Yang, M. Tomizuka, and C. Choi, “Evolvegraph: Multi-agent trajectory prediction with dynamic relational reasoning,” *arXiv preprint arXiv:2003.13924*, 2020.
- [210] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio, “Generating sentences from a continuous space,” *arXiv preprint arXiv:1511.06349*, 2015.

- [211] Z. Lin, G. I. Winata, P. Xu, Z. Liu, and P. Fung, “Variational transformers for diverse response generation,” *arXiv preprint arXiv:2003.12738*, 2020.
- [212] T. Wang and X. Wan, “T-cvae: Transformer-based conditioned variational autoencoder for story completion,” in *IJCAI*, 2019.
- [213] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.
- [214] Q. Kong, Y. Xu, W. Wang, and M. D. Plumbley, “Sound event detection of weakly labelled data with cnn-transformer and automatic threshold optimization,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2450–2460, 2020.
- [215] S. Becker, R. Hug, W. Hubner, and M. Arens, “Red: A simple but effective baseline predictor for the trajnet benchmark,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 0–0.
- [216] Y. Kant, D. Batra, P. Anderson, A. Schwing, D. Parikh, J. Lu, and H. Agrawal, “Spatially aware multimodal transformers for textvqa,” in *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IX 16*. Springer, 2020, pp. 715–732.
- [217] Z. Yang, Y. Lu, J. Wang, X. Yin, D. Florencio, L. Wang, C. Zhang, L. Zhang, and J. Luo, “Tap: Text-aware pre-training for text-vqa and text-caption,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 8751–8761.
- [218] C. Yu, X. Ma, J. Ren, H. Zhao, and S. Yi, “Spatio-temporal graph transformer networks for pedestrian trajectory prediction,” in *ECCV*, 2020.
- [219] F. Giuliari, I. Hasan, M. Cristani, and F. Galasso, “Transformer networks for trajectory forecasting,” *2020 25th International Conference on Pattern Recognition (ICPR)*, pp. 10 335–10 342, 2021.
- [220] K. D. Kusano, J. M. Scanlon, Y. Chen, T. L. McMurphy, R. Chen, T. Gode, and T. Victor, “Comparison of waymo rider-only crash data to human benchmarks at 7.1 million miles,” *arXiv preprint arXiv.2312.12675*, 2023.
- [221] J. Lee, Y. Lee, J. Kim, A. R. Kosiorek, S. Choi, and Y. W. Teh, “Set transformer: A framework for attention-based permutation-invariant neural networks,” in *Proceedings of the International Conference on Machine Learning (ICML)*, 2019.

- [222] A. Seff, B. Cera, D. Chen, M. Ng, A. Zhou, N. Nayakanti, K. S. Refaat, R. Al-Rfou, and B. Sapp, “Motionlm: Multi-agent motion forecasting as language modeling,” in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.
- [223] S. Ettinger, S. Cheng, B. Caine, C. Liu, H. Zhao, S. Pradhan, Y. Chai, B. Sapp, C. R. Qi, Y. Zhou, Z. Yang, A. Chouard, P. Sun, J. Ngiam, V. Vasudevan, A. McCauley, J. Shlens, and D. Anguelov, “Large scale interactive motion forecasting for autonomous driving : The waymo open motion dataset,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision, (ICCV)*, 2021.
- [224] L. Zhang, Z. Peng, Q. Li, and B. Zhou, “CAT: closed-loop adversarial training for safe end-to-end driving,” in *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, ser. Proceedings of Machine Learning Research, vol. 229. PMLR, 2023, pp. 2357–2372.
- [225] T. Dao, D. Y. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2022.
- [226] J. M. Scanlon, K. D. Kusano, T. Daniel, C. Alderson, A. Ogle, and T. Victor, “Waymo simulated driving behavior in reconstructed fatal crashes within an autonomous vehicle operating domain,” *Accident Analysis & Prevention*, 2021.
- [227] Y. Li, W. Yuan, S. Zhang, W. Yan, Q. Shen, C. Wang, and M. Yang, “Choose your simulator wisely: A review on open-source simulators for autonomous driving,” *IEEE Trans. Intell. Veh.*, 2024.
- [228] FHWA, “Average annual miles per driver by age group,” <https://www.fhwa.dot.gov/ohim/onh00/bar8.htm>, 2022, accessed: 2024-09-17.
- [229] M. Cusumano-Towner, D. Hafner, A. Hertzberg, B. Huval, A. Petrenko, E. Vinitzky, E. Wijmans, T. Killian, S. Bowers, O. Sener *et al.*, “Robust autonomy emerges from self-play,” *arXiv preprint arXiv:2502.03349*, 2025.
- [230] OpenStreetMap contributors, “Planet dump retrieved from <https://planet.osm.org>,” 2026, <https://www.openstreetmap.org>.
- [231] N. Kalra and S. M. Paddock, “Driving to safety: How many miles of driving would it take to demonstrate autonomous vehicle reliability?” *Transportation Research Part A: Policy and Practice*, vol. 94, pp. 182–193, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0965856416302129>

- [232] Waymo Team, “Off road, but not offline: How simulation helps advance our waymo driver,” Waypoint — The official Waymo blog, Apr. 2020, accessed: 2025-10-20. [Online]. Available: <https://waymo.com/blog/2020/04/off-road-but-not-offline--simulation27>
- [233] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, “Hierarchical text-conditional image generation with CLIP latents,” *arXiv*, 2022.
- [234] J. Abramson, J. Adler, J. Dunger, R. Evans, T. Green, A. Pritzel, O. Ronneberger, L. Willmore, A. J. Ballard, J. Bambrick, S. W. Bodenstein, D. A. Evans, C.-C. Hung, M. O’Neill, D. Reiman, K. Tunyasuvunakool, Z. Wu, A. Žemgulytė, E. Arvaniti, C. Beattie, O. Bertolli, A. Bridgland, A. Cherepanov, M. Congreve, A. I. Cowen-Rivers, A. Cowie, M. Figurnov, F. B. Fuchs, H. Gladman, R. Jain, Y. A. Khan, C. M. R. Low, K. Perlin, A. Potapenko, P. Savy, S. Singh, A. Stecula, A. Thillaisundaram, C. Tong, S. Yakneen, E. D. Zhong, M. Zielinski, A. Židek, V. Bapst, P. Kohli, M. Jaderberg, D. Hassabis, and J. M. Jumper, “Accurate structure prediction of biomolecular interactions with alphafold 3,” *Nature*, 2024.
- [235] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, “beta-VAE: Learning basic visual concepts with a constrained variational framework,” in *ICLR*, 2017.
- [236] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” in *NeurIPS*, 2017.
- [237] M. Liang, B. Yang, R. Hu, Y. Chen, R. Liao, S. Feng, and R. Urtasun, “Learning lane graph representations for motion forecasting,” in *ECCV*, 2020.
- [238] D. P. Kingma and M. Welling, “Auto-encoding variational bayes,” in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2014.
- [239] W. Peebles and S. Xie, “Scalable diffusion models with transformers,” in *ICCV*. IEEE, 2023.
- [240] J. Phillion, X. B. Peng, and S. Fidler, “Trajenglish: Traffic modeling as next-token prediction,” in *ICLR*, 2024.
- [241] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

- [242] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, “Mastering the game of go without human knowledge,” *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
- [243] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray *et al.*, “Learning dexterous in-hand manipulation,” *The International Journal of Robotics Research*, vol. 39, no. 1, pp. 3–20, 2020.
- [244] D. Silver, S. Singh, D. Precup, and R. S. Sutton, “Reward is enough,” *Artificial Intelligence*, p. 103535, 2021.
- [245] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh, “On the expressivity of markov reward,” *Advances in Neural Information Processing Systems*, vol. 34, 2021.
- [246] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané, “Concrete problems in ai safety,” *arXiv preprint arXiv:1606.06565*, 2016.
- [247] G. Dulac-Arnold, D. Mankowitz, and T. Hester, “Challenges of real-world reinforcement learning,” *arXiv preprint arXiv:1904.12901*, 2019.
- [248] M. Turchetta, A. Kolobov, S. Shah, A. Krause, and A. Agarwal, “Safe reinforcement learning via curriculum induction,” *arXiv preprint arXiv:2006.12136*, 2020.
- [249] E. Marchesini, D. Corsi, and A. Farinelli, “Exploring safer behaviors for deep reinforcement learning.” AAAI, 2022.
- [250] R. Girgis, R. de Schaetzen, L. Rowe, A. Robitaille, C. Pal, and L. Paull, “Constrained group relative policy optimization,” *arXiv preprint arXiv:2602.05863*, 2026.
- [251] H. Uzawa, K. Anow, and L. Hurwicz, “Studies in linear and nonlinear programming,” 1958.
- [252] B. Polyak, “Iterative methods using lagrange multipliers for solving extremal problems with constraints of the equation type,” *USSR Computational Mathematics and Mathematical Physics*, vol. 10, no. 5, pp. 42–52, 1970.
- [253] G. Korpelevich, “An extragradient method for finding saddle points and for other problems,” 1976.

- [254] T. Lin, C. Jin, and M. Jordan, “On gradient descent ascent for nonconvex-concave minimax problems,” in *International Conference on Machine Learning*. PMLR, 2020, pp. 6083–6093.
- [255] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” *Advances in neural information processing systems*, vol. 27, 2014.
- [256] I. Kostrikov, K. K. Agrawal, D. Dwibedi, S. Levine, and J. Tompson, “Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning,” *arXiv preprint arXiv:1809.02925*, 2018.
- [257] W. Cao, M. Hallgarten, T. Li, D. Dauner, X. Gu, C. Wang, Y. Miron, M. Aiello, H. Li, I. Gilitschenski, B. Ivanovic, M. Pavone, A. Geiger, and K. Chitta, “Pseudo-simulation for autonomous driving,” in *Conference on Robot Learning (CoRL)*, 2025.
- [258] O. Contributors, “Openscene: The largest up-to-date 3d occupancy prediction benchmark in autonomous driving,” <https://github.com/OpenDriveLab/OpenScene>, 2023.
- [259] D. Dauner, M. Hallgarten, T. Li, X. Weng, Z. Huang, Z. Yang, H. Li, I. Gilitschenski, B. Ivanovic, M. Pavone, A. Geiger, and K. Chitta, “Navsim: Data-driven non-reactive autonomous vehicle simulation and benchmarking,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- [260] T. Li, Y. Qiu, Z. Wu, C. Lindström, P. Su, M. Nießner, and H. Li, “Mtgs: Multi-traversal gaussian splatting,” *arXiv preprint arXiv:2503.12552*, 2025.
- [261] D. Dauner, M. Hallgarten, A. Geiger, and K. Chitta, “Parting with misconceptions about learning-based vehicle motion planning,” in *Conference on Robot Learning*. PMLR, 2023, pp. 1268–1281.
- [262] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang, H. Zhong, Y. Zhu, M. Yang, Z. Li, J. Wan, P. Wang, W. Ding, Z. Fu, Y. Xu, J. Ye, X. Zhang, T. Xie, Z. Cheng, H. Zhang, Z. Yang, H. Xu, and J. Lin, “Qwen2.5-vl technical report,” *ArXiv*, vol. abs/2502.13923, 2025. [Online]. Available: <https://api.semanticscholar.org/CorpusID:276449796>
- [263] H. Arai, K. Miwa, K. Sasaki, Y. Yamaguchi, K. Watanabe, S. Aoki, and I. Yamamoto, “Covla: Comprehensive vision-language-action dataset for autonomous driving,” *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 1933–1943, 2024. [Online]. Available: <https://api.semanticscholar.org/CorpusID:271909725>

- [264] L. Rowe, R. de Schaetzen, R. Girgis, C. Pal, and L. Paull, “Poutine: Vision-language-trajectory pre-training and reinforcement learning post-training enable robust end-to-end autonomous driving,” *arXiv preprint arXiv:2506.11234*, 2025.
- [265] B. Jaeger, D. Dauner, J. Beißwenger, S. Gerstenecker, K. Chitta, and A. Geiger, “Carl: Learning scalable planning policies with simple rewards,” *arXiv preprint arXiv:2504.17838*, 2025.
- [266] A. Gosselin, G. Y. Luo, L. Lara, F. Golemo, D. Nowrouzezahrai, L. Paull, A. Jolicoeur-Martineau, and C. Pal, “Ctrl-crash: Controllable diffusion for realistic car crashes,” *arXiv preprint arXiv:2506.00227*, 2025.
- [267] Z. Zhou, J. Wang, Y.-H. Li, and Y.-K. Huang, “Query-centric trajectory prediction,” *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 17 863–17 873, 2023. [Online]. Available: <https://api.semanticscholar.org/CorpusID:259359908>
- [268] N. Nayakanti, R. Al-Rfou, A. Zhou, K. Goel, K. S. Refaat, and B. Sapp, “Wayformer: Motion forecasting via simple & efficient attention networks,” *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2980–2987, 2022. [Online]. Available: <https://api.semanticscholar.org/CorpusID:250493056>
- [269] Y. Bengio, M. Cohen, D. Fornasiere, J. Ghosn, P. Greiner, M. MacDermott, S. Mindermann, A. Oberman, J. Richardson, O. Richardson *et al.*, “Superintelligent agents pose catastrophic risks: Can scientist ai offer a safer path?” *arXiv preprint arXiv:2502.15657*, 2025.
- [270] L. Gao, Y. Xu, W. Koch, S. Ruffino, L. Rowe, B. Chalaki, D. Rivkin, J. Ost, R. Girgis, M. Bijelic *et al.*, “Scenariocontrol: Vision-language controllable vectorized latent scenario generation,” *arXiv preprint arXiv:2604.17147*, 2026.
- [271] J. Garcia and F. Fernández, “A comprehensive survey on safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.
- [272] P. Karkus, M. Igl, Y. Chen, K. Chitta, J. Packer, B. Douillard, R. Tian, A. Naumann, G. Garcia-Cobo, S. Tan *et al.*, “Beyond behavior cloning in autonomous driving: a survey of closed-loop training techniques,” *Authorea Preprints*, 2025.
- [273] S. Kobayashi, Y. Schimpf, M. Schlegel, A. Steger, M. Wolczyk, J. von Oswald, N. Scherrer, K. Maile, G. Lajoie, B. A. Richards *et al.*, “Emergent temporal abstractions

- in autoregressive models enable hierarchical reinforcement learning,” *arXiv preprint arXiv:2512.20605*, 2025.
- [274] B. Terver, T.-Y. Yang, J. Ponce, A. Bardes, and Y. LeCun, “What drives success in physical planning with joint-embedding predictive world models?” *arXiv preprint arXiv:2512.24497*, 2025.
- [275] M. Destrade, O. Bounou, Q. L. Lidec, J. Ponce, and Y. LeCun, “Value-guided action planning with jepa world models,” *arXiv preprint arXiv:2601.00844*, 2025.
- [276] D. J. Yeong, G. A. Velasco-Hernández, J. Barry, and J. Walsh, “Sensor and sensor fusion technology in autonomous vehicles: A review,” *Sensors (Basel, Switzerland)*, vol. 21, 2021.
- [277] A. Ibrahim, B. Thérien, K. Gupta, M. L. Richter, Q. Anthony, T. Lesort, E. Belilovsky, and I. Rish, “Simple and scalable strategies to continually pre-train large language models,” *arXiv preprint arXiv:2403.08763*, 2024.
- [278] Y. Hu, S. Chai, Z. Yang, J. Qian, K. Li, W. Shao, H. Zhang, W. Xu, and Q. Liu, “Solving motion planning tasks with a scalable generative model,” in *ECCV*, 2024.
- [279] Z. Zhou, J. Wang, Y. Li, and Y. Huang, “Query-centric trajectory prediction,” in *CVPR*, 2023.
- [280] A. Q. Nichol and P. Dhariwal, “Improved denoising diffusion probabilistic models,” in *ICML*, 2021.
- [281] A. Ajay, Y. Du, A. Gupta, J. B. Tenenbaum, T. S. Jaakkola, and P. Agrawal, “Is conditional generative modeling all you need for decision making?” in *ICLR*, 2023.
- [282] H. Chu, D. Li, D. Acuna, A. Kar, M. Shugrina, X. Wei, M.-Y. Liu, A. Torralba, and S. Fidler, “Neural turtle graphics for modeling city road layouts,” in *ICCV*, 2019.
- [283] A. Y. Ng, D. Harada, and S. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *Icml*, vol. 99, 1999, pp. 278–287.
- [284] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
- [285] E. Beeching, M. Peter, P. Marcotte, J. Debangoye, O. Simonin, J. Romoff, and C. Wolf, “Graph augmented deep reinforcement learning in the gamerland3d environment,” *arXiv preprint arXiv:2112.11731*, 2021.

APPENDIX A LATENT VARIABLE SEQUENTIAL SET TRANSFORMERS & AUTOBOTS

A.1 Permutation Equivariance of the Model

In this section, we prove the permutation equivariance of the Latent Variable Sequential Set Transformer with respect to the dimension M . To do so, we analyze the model’s response to permutation along this dimension of the input tensor. This tensor, $\mathbf{X}$, is of dimension (K, M, t) , where $K, M, t \in \mathbb{N}$. The dimensions t and K are ordered, i.e. if we index into the tensor along dimension M (we denote this operation as $\mathbf{X}_m$ where $m \in \{1, \dots, M\}$) then we retrieve a (K, t) -dimensional matrix. However, the M dimension is unordered. This implies that when we index by another dimension, for example t , we retrieve an invertible multiset of vectors, denoted $\mathbb{X}_\tau = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$ where $\tau \in \{1, \dots, t\}$ and $\mathbf{x}_m \in \mathbb{X}_\tau$ are K -dimensional.

Our proof that our model is permutation equivariant on M demonstrates a clear difference with the properties of similar models. For example, the model proposed in Set Transformers [49] features a permutation equivariant encoder combined with a permutation *invariant* decoder. And while Deep Sets [205] provides a theoretical framework for designing neural network models that operating on sets, they do not directly address the processing of heterogeneous data structures as we do here.

The rest of the section will proceed in the following manner:

1. Provide definitions and lemmas of the mathematical structures we will use in our proofs.
2. Show the permutation equivariance of the AutoBot encoder on M .
3. Show the permutation equivariance of the AutoBot decoder on M .

For clarity, we re-state the motivation for proving permutation equivariance in this particular model. In tasks like motion prediction, we may not be able to uniquely identify moving objects (perhaps due to occlusion or similar visual features). Therefore, we may wish to represent these objects as a collection of states evolving over time. Readers may benefit from this perspective on $\mathbf{X}$ as a time-evolving (t) collection of M objects with K attributes. We described $\mathbb{X}_\tau$ as a multiset for generality, as it permits repeated elements. However, in motion prediction tasks it might be safe to assume that no two distinct objects will share the same attribute vector. By proving that our model is equivariant with respect to M , we ensure that it will not waste capacity on spurious order-specific features. The inductive bias

for this type of task differs from other settings like machine translation where word order is meaningful.

A.1.1 Definitions and Lemmas

A permutation equivariant function is one that achieves the same result if you permute the set then apply the function to the permuted set, as you would achieve if you had applied the function to the set then applied the same permutation to the output set. We provide a formal description in Def A.1.1.

Definition A.1.1 (Permutation Equivariance). Let $\mathbb{X}$ be the set of n vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$, and S_n the group of permutations of n elements $\pi : \mathbb{X} \rightarrow \mathbb{X}$. Suppose we have a function $f : \mathbb{X} \rightarrow \mathbb{X}$, we call f **Permutation Equivariant** iff $f(\pi(\mathbb{X})) = \pi(f(\mathbb{X}))$.

One of the central components of the Sequential Set Transformer is the Multi-Head Self Attention function (MHSA : $\mathbb{X} \rightarrow \mathbb{X}$) originally introduced in [48]. MHSA computes an updated representation for one input vector in a set of input vectors by processing its relation with a collection of other vectors. Without the explicit addition of positional information to the content of these vectors, this operation is *invariant* to input order. MHSA is a content-based attention mechanism, and updates the contents of a *single* input vector based on the contents of the others. However, if we define $\mathbf{f}_{\text{MHSA}}$ as the function applying MHSA to update *each* input vector, then we have created a permutation equivariant function, described in Lemma A.1.2.

Lemma A.1.2 ($\mathbf{f}_{\text{MAB}}$ is Permutation Equivariant on M). *Let $\mathbb{X}$ be a set of D -dimensional vectors $\{\mathbf{x}_1, \dots, \mathbf{x}_M\}$, and $\mathbf{f}_{\text{MAB}} : \mathbb{R}^{D \times M} \rightarrow \mathbb{R}^{D \times M}$ be a function that applies MAB (as defined in Equation 4.1 of the main text) to each $\mathbf{x}_m \in \mathbb{X}$. Then $\mathbf{f}_{\text{MAB}}(\mathbb{X}_\tau)$ is permutation equivariant because $\forall \pi \in S_m, \mathbf{f}_{\text{MAB}}(\pi(\mathbb{X})) = \pi(\mathbf{f}_{\text{MAB}}(\mathbb{X}))$.*

Having discussed the preliminaries, we now show the permutation equivariance of AutoBot’s encoder.

A.1.2 AutoBot Encoder Properties

Theorem A.1.3 (AutoBot Encoder is Permutation Equivariant on M). *Let $\mathbf{X}$ be a tensor of dimension (K, M, t) where $K, M, t \in \mathbb{N}$, and where selecting on t retrieves an invertible multiset of sequences, denoted $\mathbb{X}_\tau = \{\mathbf{x}_1, \dots, \mathbf{x}_M\}$, where $\tau \in \{1, \dots, t\}$ and $\mathbf{x}_m \in \mathbb{X}_\tau$ are K -dimensional. The AutoBot encoder, as defined in Section 4.4.1, is permutation equivariant on M .*

Proof. What follows is a direct proof of the permutation equivariance of the AutoBot encoder. The encoder is a composition of three functions, $\text{rFFN}: \mathbb{R}^{K \times t} \rightarrow \mathbb{R}^{d_K \times t}$, $\mathbf{f}_{\text{MAB}}: \mathbb{R}^{d_k \times M} \rightarrow \mathbb{R}^{d_k \times M}$, and $\mathbf{f}_{\text{MAB}}: \mathbb{R}^{d_K \times t \times M} \rightarrow \mathbb{R}^{d_K \times t \times M}$.

First, we show that the embedding function rFFN is equivariant on M . Let rFFN represent the application of a function $e: \mathbb{R}^K \rightarrow \mathbb{R}^{d_K}$ to each element of a set $\mathbb{X}_\tau$ where $\tau \in \{1, \dots, t\}$. Then rFFN is equivariant with respect to M by Definition A.1.1. Specifically, we can see that the function rFFN satisfies the equation $\forall \pi \in S_M, \text{rFFN}(\pi(\mathbb{X}_\tau)) = \pi(\text{rFFN}(\mathbb{X}_\tau))$. In the AutoBot encoder, the function rFFN is applied to each set $\mathbb{X}_\tau$, and because it is permutation equivariant on M for each, the application to each set represents a permutation equivariant transformation of the entire tensor.

Next, we show that the function $\mathbf{f}_{\text{MAB}}$, which applies a Multi-Head Attention Block (MAB) to each set, $(\mathbb{X}_1, \dots, \mathbb{X}_t)$, is equivariant on M . $\mathbf{f}_{\text{MAB}}$ is permutation equivariant with respect to the M dimension of $\mathbf{X}$ because MAB is permutation equivariant on each set $\mathbb{X}_\tau$ by Lemma A.1.2.

Finally, we define a function $\mathbf{f}_{\text{MAB}}$ which applies a positional encoding along the time dimension of $\mathbf{X}$ then a Multi-head Attention Block (MAB) to each matrix in $\mathbf{X}$, $\{\mathbf{X}_1, \dots, \mathbf{X}_M\}$. $\mathbf{f}_{\text{MAB}}$ is permutation equivariant with respect to the M dimension of $\mathbf{X}$ because the collection of matrices $\{\mathbf{X}_1, \dots, \mathbf{X}_M\}$ is an unordered set, and the uniform application of a function to each element of this set satisfies permutation equivariance.

A composition of equivariant functions is equivariant, so the encoder is equivariant on M . $\square$

A.1.3 AutoBot Decoder Properties

We now provide a direct proof the the permutation equivariance of the AutoBot decoder on M . The decoder is an auto-regressive function initialized with a seed set $\mathbb{X}_t$ containing M vectors of dimension d_k , each concatenated with a vector of latent variables and other context of dimension $2d_k$. For $\tau \in \{1, \dots, T - t\}$ iterations, we concatenate the output of the decoder with the seed, the previous decoder output, and the latents to produce a tensor of dimension $(3d_K, M, \tau)$. In particular, the decoder is a composition of three functions, $\text{rFFN}_{\text{dec}}: \mathbb{R}^{3d_K \times \tau} \rightarrow \mathbb{R}^{d_K \times \tau}$, $\mathbf{f}_{\text{MAB}}: \mathbb{R}^{d_k \times M} \rightarrow \mathbb{R}^{d_k \times M}$, and $\mathbf{f}_{\text{MABD}}: \mathbb{R}^{d_K \times M \times \tau} \rightarrow \mathbb{R}^{d_K \times M \times \tau}$.

Theorem A.1.4 (AutoBot Decoder is Permutation Equivariant on M). *Given an invertible multiset $\mathbb{X}_t$ with M vector-valued elements, and an auto-regressive generator iterated for $\tau \in \{1, \dots, T - t\}$, the AutoBot decoder, formed by a composition of three functions, $\text{rFFN}_{\text{dec}}: \mathbb{R}^{3d_K \times M} \rightarrow \mathbb{R}^{d_K \times M}$, $\mathbf{f}_{\text{MAB}}: \mathbb{R}^{d_k \times M} \rightarrow \mathbb{R}^{d_k \times M}$, and $\mathbf{f}_{\text{MABD}}: \mathbb{R}^{d_K \times \tau \times M} \rightarrow \mathbb{R}^{d_K \times \tau \times M}$, is equivariant on M .*

Proof. First, we must establish that the function $\text{rFFN}_{dec}: \mathbb{R}^{3d_K \times M} \rightarrow \mathbb{R}^{d_K \times M}$ is equivariant with respect to M . The AutoBot decoder applies rFFN_{dec} to each set $\mathbb{X}_\tau$ where $\tau \in \{1, \dots, T - t\}$, transforming the M vectors of dimension $3d_K \rightarrow d_K$. Because rFFN_{dec} represents a uniform application of e_{dec} to each element of the set, we can see that it satisfies the definition of permutation equivariance, specifically that $\forall \pi \in S_M, \text{rFFN}_{dec}(\pi(\mathbb{X}_\tau)) = \pi(\text{rFFN}_{dec}(\mathbb{X}_\tau))$.

Next, we take the function $\mathbf{f}_{\text{MABD}}$, which first applies a function $\text{PE}: \mathbb{R}^{d_K, M, \tau} \rightarrow \mathbb{R}^{d_K, M, \tau}$ to $\mathbf{X}$ adding position information along the temporal dimension, then applies $f_{\text{MABD}}: \mathbb{R}^{d_K \times \tau} \rightarrow \mathbb{R}^{d_K \times \tau}$ to each matrix $\{\mathbf{X}_1, \dots, \mathbf{X}_M\}$. $\mathbf{f}_{\text{MABD}}$ is permutation equivariant with respect to the M dimension of $\mathbf{X}$ because the collection of matrices $\{\mathbf{X}_1, \dots, \mathbf{X}_M\}$ is an unordered set, and the uniform application of a function to transform each element independently does not impose an order on this set.

Similar to the final step of the previous proof, we see that a function $\mathbf{f}_{\text{MAB}}$ applying a multi-head self-attention block (MAB) to each set $\{\mathbb{X}_{t+1}, \dots, \mathbb{X}_\tau\}$ is equivariant on M because MAB is permutation equivariant on each set (see Lemma A.1.2).

The composition of equivariant functions is equivariant, so the decoder is equivariant. $\square$

A.2 Model Details

A.2.1 Ego-Centric Forecasting

AutoBots can be used to predict the future trajectories of all agents in a scene. However, in some settings we may wish to forecast the future trajectory of only a single agent in the set. We call this variant ‘‘AutoBot-Ego’’. Referring back to the encoder presented in Section 4.4.1, instead of propagating the encoding of all elements in the set, we can instead select the state of that element produced by the intra-set attention, $\mathbf{s} \in \mathbb{S}_\tau$. With this, our encoder proceeds with the rest of the computation identically to produce the tensor $\mathbf{C}_{1:t}^m$, which is an encoding of the ego element’s history conditioned on the past sequence of all elements.

AutoBot-Ego’s decoder proceeds as presented in section 4.4.2, with one exception. Since AutoBot-Ego only generates the future trajectory of an ego element, we do not perform intra-set attention in the decoder. As a consequence of only dealing with a single ego-agent in the future generation, we do not repeat the seed parameter matrices $\mathbf{Q}_i$ across the M dimension, where $i \in \{1, \dots, c\}$. The objective function is updated to compute the likelihood of one element’s future given the entire input sequence. That is, in AutoBot-Ego, $\mathcal{Y} = \mathbf{X}_{t+1:T}^m$.

A.2.2 Expectation Maximization Details

Our model also computes the distribution $P(Z|\mathbf{X}_{1:t})$ of discrete random variables. To achieve this, we employ c learnable vectors (one for each mode) concatenated into the matrix $\mathbf{P} \in \mathbb{R}^{(d_K, c)}$, which behave like seeds to compute distribution over modes prior to observing the future (these seed parameters should not be confused with the decoder seed parameters $\mathbf{Q}$). The distribution is generated by performing the computations

$$p(Z|\mathbf{X}_{1:t}) = \text{softmax}(\text{rLin}(\mathbf{F})), \quad \text{where} \quad \mathbf{F} = \text{MABD}(\mathbf{P}_{1:c}, \mathbf{C}),$$

where rLin is a row-wise linear projection layer to a vector of size c .

A.2.3 Context Encoding Details

In the nuScenes dataset, we are provided with a birds-eye-view 128×128 RGB image of the road network, as shown in Figure A.1. In the TrajNet dataset, we are also provided with a birds-eye-view of the scene, as shown in Figure A.2. We process this additional context using a 4 layer convolutional neural network (CNN) which encodes the map information into a volume of size $7 \times 7 \times (7 * c)$ where c is the number of modes. We apply a 2D dropout layer with a rate of 0.1 on this output volume before processing it further. As our model employs discrete latent variables, we found it helpful to divide this volume equally among all c modes, where each mode receives a flattened version of the $7 \times 7 \times 7$ volume. As described in Section 4.4.2, this context is copied across the M and T dimensions, and concatenated with the decoder seed parameters during the sequence generation process. Intuitively, each mode’s generated trajectory is conditioned on a different representation of the context.

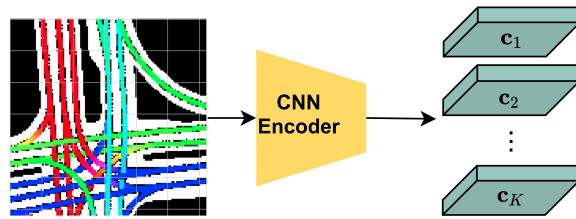


Figure A.1 **AutoBots Context Encoder.** Example of the birds-eye-view road network provided by the Nuscenes dataset. Our CNN encoder produces a volume which we partition equally into K modes. This allows each generated trajectory to be conditioned on a different representation of the input image.



Figure A.2 **TrajNet Map Example.** Example birds-eye-view map provided by the TrajNet benchmark. The footage was captured by a drone and pedestrians and bicycles are freely moving through the scene.

Parameter	Description	Toy	OmniGlott	Nuscenes	Argoverse	TrajNet	TrajNet++
d_k	Hidden dimension throughout all parts of the model.	64	128	128	128	128	128
Learning Rate	Learning rate used in Adam Optimizer.	1e-4	5e-4	7.5e-4	7.5e-4	7.5e-4	5e-4
Batch size	Batch size used during training.	6	64	64	64	64	64
c	Number of discrete latent variables.	10	4	10	6	5	6
λ_e	Entropy regularization strength.	3.0	1.0	30.0	30.0	5.0	30.0
Dropout	Amount of dropout used in MHSA functions.	0.0	0.2	0.1	0.1	0.3	0.05
L_{enc}	Number of stacked social/Temporal blocks in the encoder.	1	1	2	2	2	2
L_{dec}	Number of stacked social/Temporal blocks in the decoder.	1	1	2	2	2	2
H	Number of attention heads in all MHSAs.	8	8	16	16	8	16

Table A.1 Hyperparameters used for AutoBots across all four datasets.

A.2.4 Implementation and Training details

We implemented our model using the Pytorch open-source framework. Table A.1 shows the values of parameters used in our experiments across the different datasets. The MAB encoder and decoder blocks in all parts of AutoBots use dropout. We train the model using the Adam optimizer with an initial learning rate. For the Nuscenes and TrajNet++ datasets, we anneal the learning rate every 10 epochs for the first 20 epochs by a factor of 2, followed by annealing it by a factor of 1.33 every 10 epochs for the next 30 epochs. For the Argoverse dataset, we anneal the learning rate every 5 epochs by a factor 2 for the first 20 epochs. In all datasets, we found it helpful to clip the gradients to a maximum magnitude of 5.0. For the Nuscenes experiments, our model takes approximately 80 epochs to converge, which corresponds to approximately 3 hours of compute time on a single Nvidia Geforce GTX 1080Ti GPU, using approximately 2 GB of VRAM. We used mirroring of all trajectories and the map information as a data augmentation method since the dataset’s scenes originate from Singapore and Boston. For Argoverse experiments, our model takes approximately 30 epochs to converge, which corresponds to approximately 10 hours on the same resources. In order to improve the performance, we use other agents in the certain scenes as an ego-agent, effectively doubling the size of the dataset. We chose agents that move by at least 10 meters. This, in-turn, increases the training time to approximately 24 hours. For the TrajNet, TrajNet++ and Omniglot datasets, our model takes approximately 100 epochs which corresponds to a maximum 2 hours of compute time on the same resources.

A.2.5 Performance - Comparing Seed Parameter Decoder with Autoregressive Decoder

One of the main architectural contributions of our work compared to our contemporaries is the use of seed parameters in the decoder network to allow for one-shot prediction of the entire future trajectory (compared to the established way of rolling out the model autoregressively timestep by timestep). To this end, we created an ablation of our model that predicts the future timesteps autoregressively and we compare its performance to ours across different variables (number of agents, input sequence length, output sequence length, and number of modes). Figure A.3 shows the inference rate (i.e. full trajectory predictions per second) of AutoBot-Ego (top) and AutoBot (bottom). We can see that in the single-agent case (AutoBot-Ego), both the seed parameter version of the model as well as the autoregressive ablation are able to operate in real-time (≥ 30 FPS), save for when using a prediction horizon longer than 20 steps. We chose 30 FPS as a real-time indicator line since most cameras used in self-driving cars output at least 30 FPS [276]; Lidar data is usually slower to retrieve

at 5-20 FPS. The performance difference is more obvious when comparing AutoBot to its ablation. The autoregressive model is barely able to perform at 30 FPS when the number of agents is low, the number of input steps is very low, the predicted future steps are under 20, and the number of modes is low.

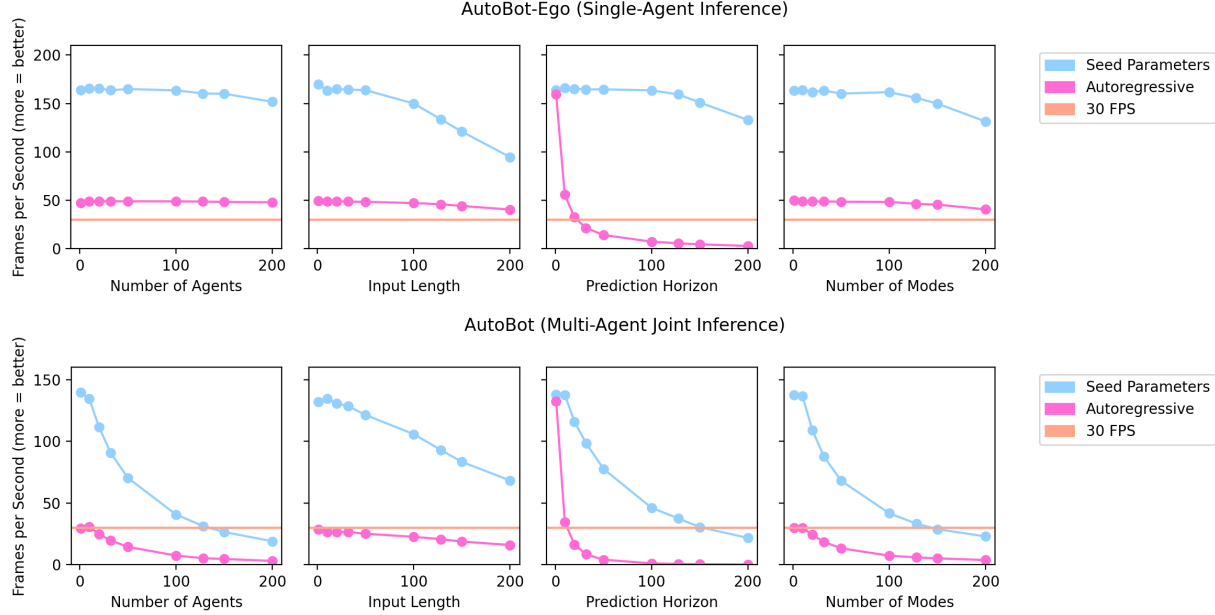


Figure A.3 **AutoBots's Inference Rate** To show the impact of using a seed parameter decoder as opposed to an autoregressive decoder, we compare the inference rate of AutoBots with its autoregressive but otherwise identical ablation. We plot how the inference rate varies with increasing size of input/output variables. From left-to-right, these variables correspond to 1) number of agents, 2) number of observable timesteps for each agent, 3) prediction horizon, and 4) number of modes. The top row corresponds to AutoBot-Ego while the bottom row corresponds to AutoBot.

A.3 AutoBots Additional Results

A.3.1 Ablation Study on Nuscenets

Table A.2 shows the results of ablating various components from AutoBot-Ego on the Nuscenets dataset. As we can see, the model trained without the map information performs significantly worse across all metrics. This shows that indeed AutoBot-Ego learns to attend to the map information effectively. The ‘No Latent Variable’ version of our model is a model without the latent variable, and therefore, without the decoder seed parameters. This model then becomes auto-regressive and samples from the output bivariate Gaussian distribution at every timestep. We note that this version of the model is significantly slower to train (taking approximately three times as long) due to its autoregressive nature. In order to generate ten trajectories, the generation process is repeated $c = 10$ times. This model uses the same transformer encoder and decoder blocks as described in Section 4.4. As we can see from its performance on the ADE metrics, without latent variables, the model is not capable of generating diverse enough trajectories. This is expected since the output model ϕ is now tasked with accounting for both the aleatoric and epistemic uncertainties in the process. Finally,

the ‘No Data Augmentation’ model shows that although it is helpful to augment the data with mirroring, even without it the model is still capable of producing strong performance.

AutoBot-Ego	Min ADE (5)	Min ADE (10)	Miss Rate Top-5 (2m)	Miss Rate Top-10 (2m)	Off Road Rate
No Map Information	1.88	1.35	0.72	0.58	0.22
No Latent Variable	1.77	1.34	0.76	0.64	0.07
No Data Augmentation	1.47	1.08	0.66	0.47	0.03
AutoBot-Ego	1.43	1.05	0.66	0.45	0.03

Table A.2 **NuScenes Ablation Study..** We study the performance benefits of AutoBot-Ego over various ablated counterparts.

A.3.2 Analysis of Entropy Regularization

This dataset contains the trajectories of a single particle having identical pasts but diverse futures, with different turning rates and speeds, as shown in Figure A.4 (top row). In the second row of Figure A.4, we show that our model trained without entropy regularization, $\lambda_e = 0.0$, can cover all modes. However, the short trajectory variants are all represented using only one of the modes (2nd row, 3rd column), as shown by the growth of the uncertainty ellipses as the trajectory moves forward. This shows that AutoBots learn to increase the variance of the bivariate Gaussian distribution in order to cover the space of trajectories. The third row of Figure A.4, shows the learned trajectories for this variant trained with $\lambda_e = 3.0$. As we can see, the entropy regularization penalizes the model if the uncertainty of bivariate Gaussians is too large. By pushing the variance of the output distribution to a low magnitude, we restrict the model such that the only way it can achieve high likelihood is to minimize the distance between the mean of the bivariate distributions and ground-truth trajectories.

Ablation Study - Importance of Entropy Loss Term. In Table A.3, we show an ablation study to evaluate the importance of the entropy component on training the model for nuScenes. Again, we observe that for multi-modal predictions, enforcing low entropy on the predicted bivariate distributions has a great positive impact on the results, with an increased λ_e resulting in better performance. This was observed on all evaluated metrics with the exception of the min FDE, which was poor regardless of the model. Furthermore, we can see from Table 4.1 that, in general, all prior models have a difficulty obtaining strong performance on the min FDE (1) metric, with the best value having an overall error of 8.49 meters. This may be due to the difficulty of predicting the correct mode when one only has

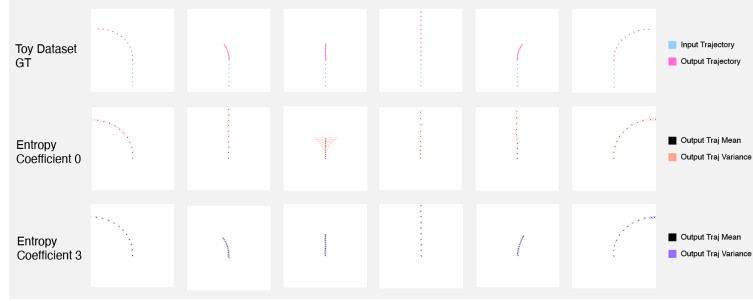


Figure A.4 **Influence of entropy loss on particle experiment.** The input trajectory (cyan, only shown in top row) is identical in all cases. The model trained without an entropy loss term (middle row) covers all modes with high variance, while the model with moderate entropy regularization (bottom row) is able to learn the modes with low variance and without overlap.

access to two seconds of the past sequence.

λ_e	Min ADE (5)	Min ADE (10)	Miss Rate Top-5 (2m)	Miss Rate Top-10 (2m)	Min FDE (1)	Off Road Rate
0	1.75	1.29	0.67	0.58	9.55	0.10
1	1.71	1.22	0.65	0.57	9.03	0.08
5	1.70	1.14	0.62	0.53	9.11	0.05
30	1.75	1.14	0.63	0.54	9.51	0.04
40	1.72	1.11	0.60	0.51	9.01	0.04

Table A.3 **NuScenes Quantitative Results by Entropy Regularization.** We study AutoBot-Ego’s performance when varying the entropy regularization term on nuScenes, and see that increased entropy improves model performance (particularly Min ADE 10 and Off Road Rate) up to a limit.

A.3.3 TrajNet++ Ablation and Qualitative results

Table A.4 shows the effect of increasing the number of social/temporal encoder and decoder layers in the general AutoBot architecture. We can see that increasing the number of layers in the decoder has a positive impact on the performance of the joint metrics. This is expected since as we increase the number of social layers, the model can use high-order interactions to model the scene. Figure A.5 shows an example scene generated by each of the model variants.

Model	Ego Agent's Min ADE(6) ($\downarrow$)	Number Of Collisions ($\downarrow$)	Scene-level Min ADE(6) ($\downarrow$)	Scene-level Min FDE(6) ($\downarrow$)
$L_{enc} = 1, L_{dec} = 1$	0.108	273	0.152	0.281
$L_{enc} = 2, L_{dec} = 1$	0.102	214	0.138	0.254
$L_{enc} = 3, L_{dec} = 1$	0.099	227	0.135	0.245
$L_{enc} = 1, L_{dec} = 2$	0.099	164	0.132	0.248
$L_{enc} = 1, L_{dec} = 3$	0.091	123	0.121	0.225
$L_{enc} = 2, L_{dec} = 2$	0.095	139	0.128	0.234
$L_{enc} = 3, L_{dec} = 3$	0.090	98	0.119	0.216

Table A.4 **TrajNet++ ablation studies for a multi-agent forecasting scenario.** We investigate the impact of the number of social/temporal layers in the encoder and the decoder.

A.3.4 TrajNet, A Pedestrian Motion Prediction Dataset

We further validate AutoBots on a pedestrian trajectory prediction dataset TrajNet [189] where we are provided with the state of all agents for the past eight timesteps and are tasked with predicting the next 12 timesteps. In this experiment, we apply our general architecture (AutoBot) to a multi-agent scene forecast situation, and ablate various the social components in the decoder only, and decoder and encoder.

Model	Number Of Collisions	Scene-level Min ADE(5)	Scene-level Min FDE(5)
AutoBot-AntiSocial	474	0.571	1.02
AutoBot-Ego	466	0.602	1.07
AutoBot	326	0.471	0.833

Table A.5 **TrajNet ablation studies for a multi-agent forecasting scenario.** We evaluate the impact of using the extra social data as a set and the actual multi-agent modelling on this context. We found that AutoBot is able to cause fewer collisions between agents compared to variants without the social attention.

In Table A.5, we present three variants of the model. The first, AutoBot-AntiSocial corresponds to a model without the intra-set attention functions in the encoder and decoder (see Figure 4.2). The second variant, AutoBot-Ego, only sees the past social context and has no social attention during the trajectory generation process. The third model corresponds to the general architecture, AutoBot. All models were trained with $c = 5$ and we report scene-level error metrics as defined by [14]. Using the social data with the multi agent modelling in the decoder greatly reduces the number of collisions between predicted agent futures. Furthermore, we can see that AutoBot significantly outperforms its ablated counterparts on

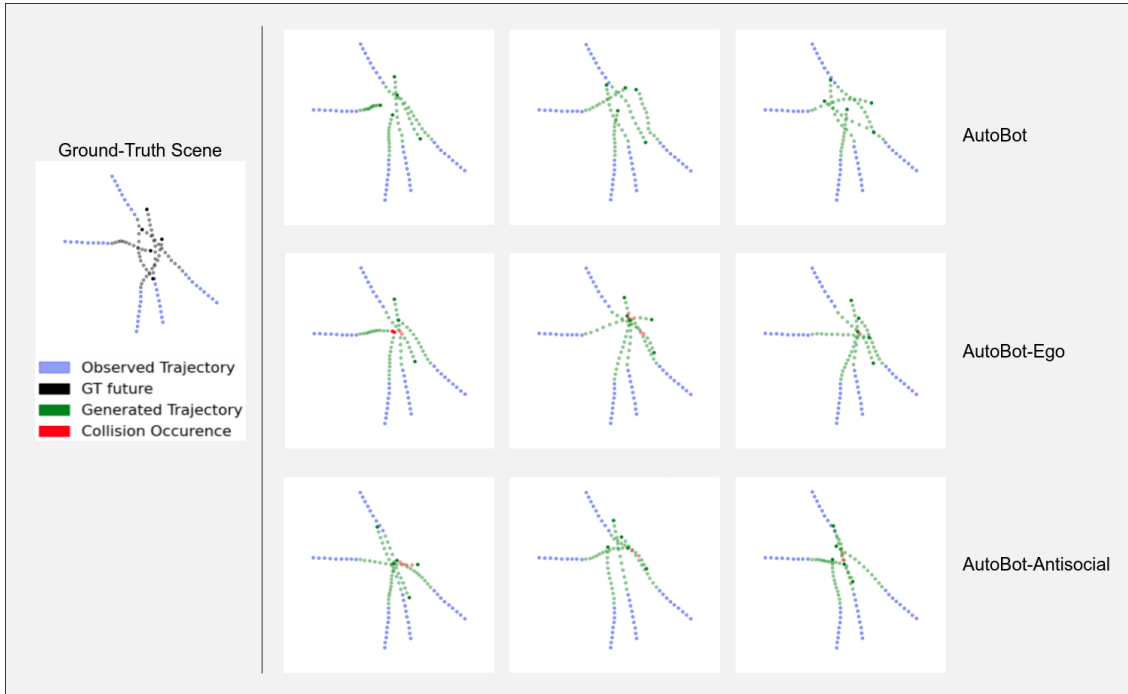


Figure A.5 **TrajNet++ qualitative results.** In this example scene, we see five agents moving towards each other with their input trajectories (blue) and their ground-truth future trajectories (black) shown in the left figure. The different rows show three of the $c = 6$ scene predictions made by each model variant. AutoBot-Ego and AutoBot-AntiSocial produce some modes containing collisions or socially inconsistent trajectories, while AutoBot, which has social attention in the decoder, achieves consistent trajectories.

generating scenes that match ground-truth, as observed by the reduced scene-level minADE and minFDE. We expect this theoretically since the ego-centric formulation makes the independence assumption that the scene’s future evolution can be decomposed as the product of the individual agent’s future motion in isolation, as remarked in [14]. AutoBot does not make this assumption as we condition between agents at every timestep.

Figure A.6 shows an example of multi-agent trajectory generation using the three different approaches for predicting the scene evolution. We present four of the five modes generated by the model for this example input scene. One can observe that the only model achieving scene-consistent predictions across all modes is AutoBot. Interestingly, we also observe that the different modes correspond to different directions of motion, highlighting the utility of using modes. In fact, the modes in AutoBot condition the entire scene future evolution, producing alternative realistic multi-agent futures. We refer the reader to Figures A.7, A.8, A.9 and A.10 where we show some additional qualitative results of AutoBots’ variants compared to AutoBot on the TrajNet dataset. These results highlight the effectiveness of the sequential set attention mechanism in the decoding process of our method and how important it is to

predict all agents in parallel compared to a 1-by-1 setup.

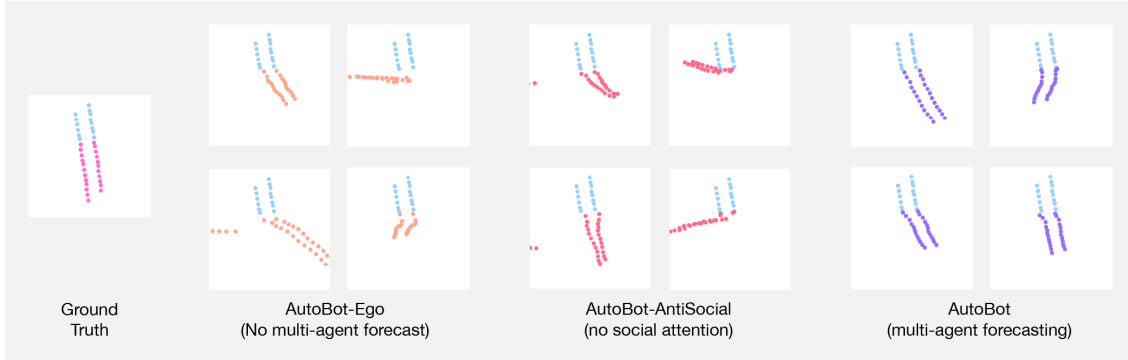


Figure A.6 **TrajNet qualitative results.** In this example, we see two agents moving together in the past (cyan) and future (pink, left). We compare how different variants of AutoBots make predictions in this social situation. AutoBot-Ego and AutoBot-AntiSocial produce some modes containing collisions or unrealistic trajectories, while AutoBot, which has intra-set attention during decoding, achieves consistent trajectories.

A.3.5 Omniglot Additional Details and Results

The LSTM baseline used in our Omniglot experiments have a hidden size of 128 and use $c = 4$ latent modes. The input is first projected into this space using an embedding layer, which is identical to the one used in AutoBot. All input sequences of sets are encoded independently using a shared LSTM encoder. During the decoding phase, we autoregressively generate the next stroke step using an LSTM decoder. In order to ensure consistency between the predicted future strokes, inspired by Social LSTM [11], the LSTM baseline employs an MHSA layer at every timestep operating on the hidden state of the different strokes making up the character. We concatenate a transformation of the one-hot vector representing the latent mode with the socially encoded hidden state at every timestep. The output model is identical to the one used AutoBot, generating a sequence of bivariate Gaussian distributions for each stroke. We performed a hyperparameter search on the learning rate and found the optimal learning rate to be $5e-4$. Furthermore, as with all our other experiments, we found it helpful to employ gradient clipping during training.

We provide additional results on the two tasks defined in Section 4.5.4. Figure A.11 shows additional successful AutoBot results on task 1 (completing multiple strokes) compared to an LSTM baseline equipped with social attention. These results highlight the effectiveness of sequential set transformers for generating consistent diverse futures given an input sequence of sets. Figure A.12 shows examples where both models fail to correctly complete the character in task 1. Figure A.13 compares AutoBot-Ego with the LSTM baseline on predicting a new

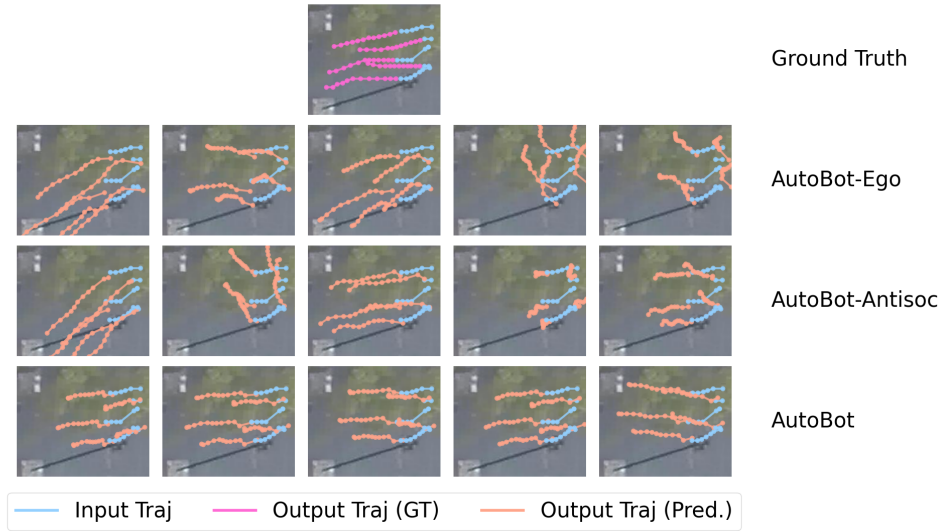


Figure A.7 **TrajNet qualitative results 1/4.** Example scene with multiple agents moving together. These trajectories are plotted over the birds-eye-view image of the scene where we zoom into interesting trajectories. We can see that only AutoBot produces trajectories that are realistic in a group setting across all modes.

stroke given the two first strokes of a character (task 2). These results highlight that although not all modes predict the correct character across all modes, all generated predictions are consistent with realistic characters (i.e., confusing “F” with “H” or “H” with “H”).

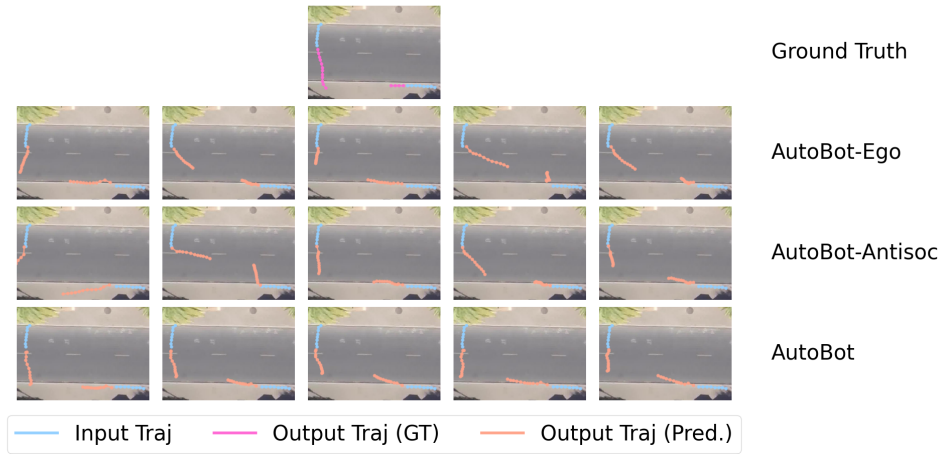


Figure A.8 **TrajNet qualitative results 2/4.** Example scene with two agents moving separately in a road setting. We want to highlight this interesting scenario where some modes of AutoBots-Solo and AutoBots-AntiSocial results in trajectories that lead into the road, while AutoBot seems to produces trajectories more in line with the ground-truth, and lead the agent to cross the road safely.

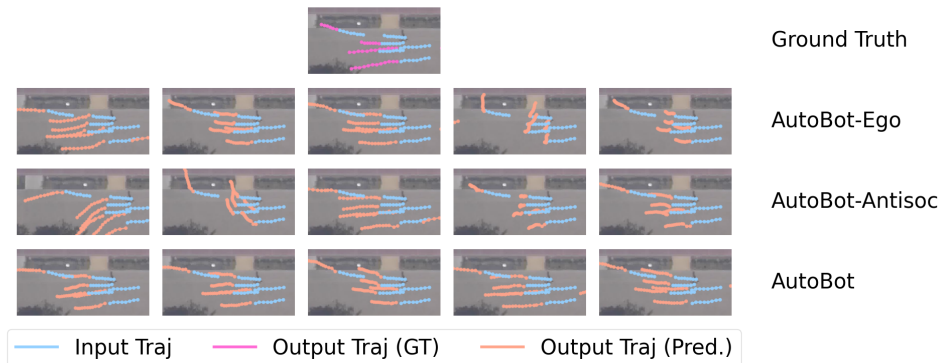


Figure A.9 **TrajNet qualitative results 3/4.** Additional example scenes with multiple agents moving together. Again, we wish to highlight the advantage of modelling the scene jointly, which is evident by the results of AutoBot.

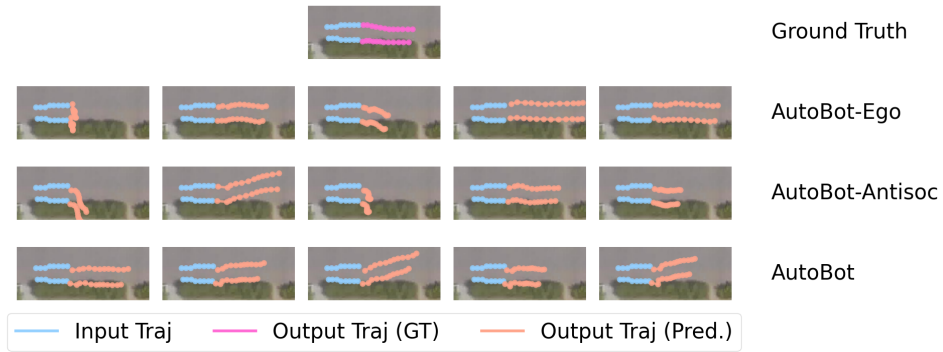


Figure A.10 **TrajNet qualitative results 4/4**. Example scenes with two agents moving together. Again, we see that AutoBot produces trajectories consistent with the scene across all modes (e.g., not crashing into the bushes) and maintains the social distance between the walking agents.

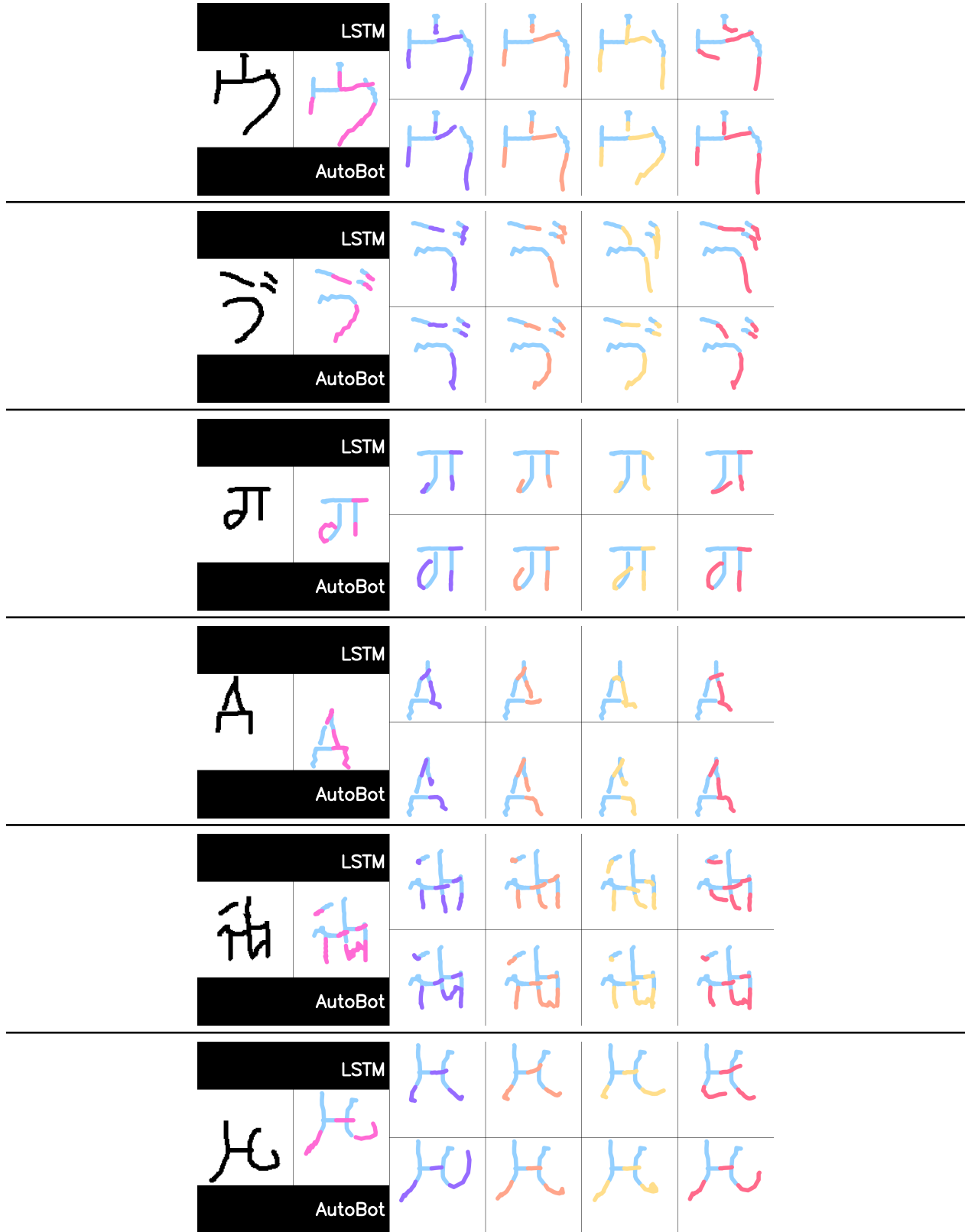


Figure A.11 **Omniglot Task 1 Additional Results.** These are some additional random characters from the Omniglot stroke completion task. We can see that AutoBot produces plausible characters on the test set, while the LSTM struggles to create legible ones.

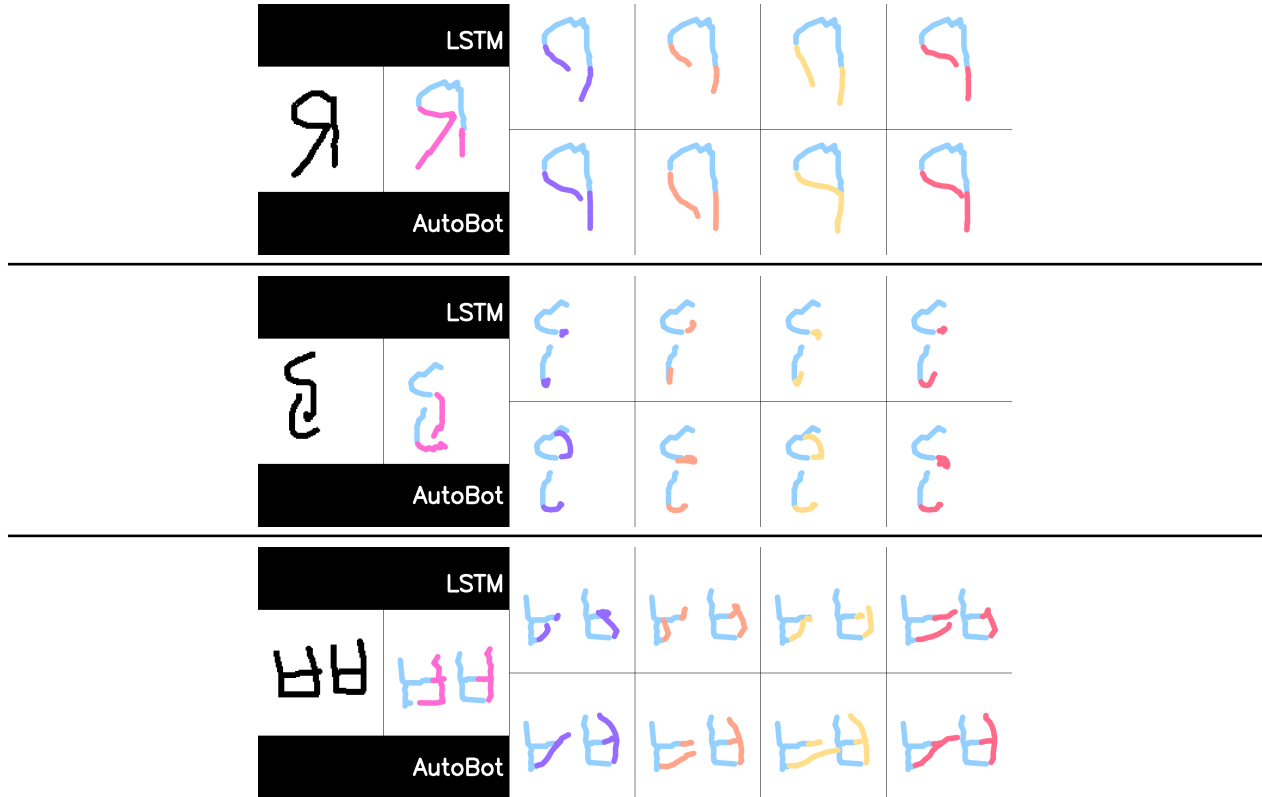


Figure A.12 **Omniglot Task 1 Failure Cases.** There were some characters where AutoBot failed to learn the correct character completion. We can currently only speculate why that is the case. Our first intuition was that this might occur when there are 90 degree or less angles in the trajectory that is to be predicted but in Fig. A.11, we can see that there are examples where this does not seem to be a problem. We believe that this might be due to a rarity of specific trajectories in the dataset but more investigation is required.

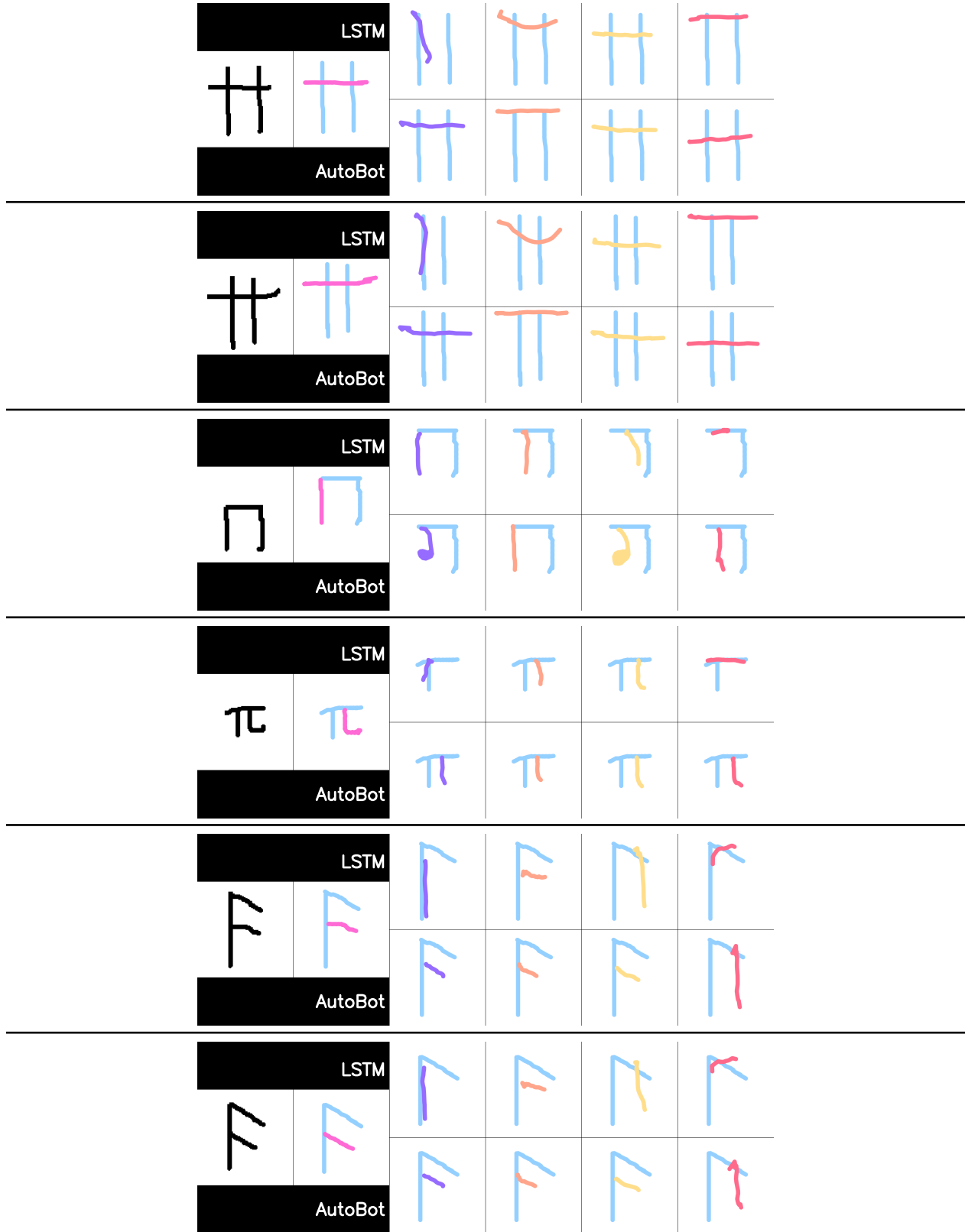


Figure A.13 **Omniglot Task 2 Additional Results.** These are some additional random characters from the Omniglot character completion task. Again, we can see that AutoBot produces plausible characters on the test set, where different modes capture plausible variations (e.g. F to H and H to PI), while the LSTM struggles to capture valid characters in its discrete latents.

APPENDIX B CONTROLLABLE BEHAVIOUR SIMULATION & CTRL-SIM

B.1 Action Sampling Algorithm

We present our action sampling procedure in Algorithm 5 for controllable behaviour generation with exponential tilting of the factorized predicted return-to-go values.

Algorithm 5 The action sampling algorithm used by CtRL-Sim to allow for factorized tilting of the exhibited behaviour.

```

1: Input:  $\{\kappa^1, \dots, \kappa^C\}$  ▷ The specified inverse temperature for each return-to-go component.
2: for  $c = 1$  to  $C$  do
3:    $G_t^c \sim p_\theta(G_t^c | s_t, s_G) \exp(\kappa^c G_t^c)$ 
4: end for
5:  $a_t \sim \pi_\theta(a_t | s_t, s_G, G_t^1, \dots, G_t^C)$ 
6: return  $a_t$ 

```

B.2 Nocturne Physics Simulator and Offline RL Dataset

B.2.1 Physics-based Nocturne Simulator

Base environment. In CtRL-Sim, we build on Nocturne [38], a lightweight 2D driving simulator derived from real trajectories in the Waymo Open Motion Dataset (WOMD) [223]. Scenes contain dynamic agents (vehicles, pedestrians, cyclists) and map context (lane boundaries/markings, traffic signs, crosswalks). Each agent is assigned a goal equal to the final waypoint of its WOMD ground-truth trajectory; if the trajectory has missing timesteps, the goal is set to the waypoint immediately preceding the first gap. By default, agents output actions that replay their exact 9s trajectories at 10Hz. In the original Nocturne setup for RL, 10 simulation steps (1s) of context are provided, and the agent must reach its goal within the next 80 steps (8s).

Extensions. We augment Nocturne with a Box2D-based physics layer to enable realistic vehicle dynamics and collisions. Control forces integrate to accelerations that determine speed and heading, with friction modelling to capture sliding and grip. We impose plausible limits on acceleration, braking, and turning radius, and permit physical vehicle–vehicle contact.

These additions allow more faithful simulation of scenarios such as emergency braking, low-friction surfaces, and multi-vehicle collisions.

B.2.2 Offline RL Dataset Collection

Actions are acceleration and steering angle. The reward is the sum of three terms: goal position, vehicle–vehicle collision, and vehicle–road-edge collision. We confirmed that running WOMB scenes through the simulator (detailed next) yields rollouts that closely reconstruct the logged trajectories (Table 6.4). Following Nocturne [38], we exclude pedestrians/bicyclists and scenes with traffic lights, yielding a training, validation, and test set containing 134150, 9678, and 2492 scenes.

Trajectory reconstruction. At timestep t , we compute $a_t = (\text{accel}, \text{steer})$ by using the inverse bicycle model from the current simulated state $\hat{s}_t$ and the next logged (GT) state s_{t+1} . We clip acceleration to $[-10, 10]$ and steering to $[-0.7, 0.7]$ rad, apply forward dynamics to obtain $\hat{s}_{t+1}$, and iterate; Table 6.4 reports reconstruction accuracy.

Rewards. For each agent, their goal is set to the final state (i.e., position, heading, and velocity) of the ground-truth logged trajectory. The goal position reward is defined by:

$$R_g(s_t, s_G) = \mathbf{1}_{\text{goal achieved}}(s_t, s_G), \quad (\text{B.1})$$

where $\text{goal achieved}(\cdot)$ is 1 if the agent ever reaches within 1 metre of the ground-truth goal, and 0 otherwise. The vehicle-vehicle collision reward is defined by:

$$R_v(s_t, \mathbb{S}_t - \{s_t\}) = -10 \times \mathbf{1}_{\text{vehicle-vehicle collision}}(s_t, \mathbb{S}_t - \{s_t\}) \quad (\text{B.2})$$

$$+ \frac{\min(\text{dist-nearest-vehicle}(s_t, \mathbb{S}_t - \{s_t\}), 15)}{15}, \quad (\text{B.3})$$

where $\text{dist-nearest-vehicle}(\cdot)$ computes the distance between the agent of interest and its nearest agent in the scene. Finally, the vehicle-road-edge collision reward is defined by:

$$R_e(s_t, m) = -10 \times \mathbf{1}_{\text{vehicle-road-edge collision}}(s_t, m) + \frac{\min(\text{dist-nearest-road-edge}(s_t, m), 5)}{5}, \quad (\text{B.4})$$

where $\text{dist-nearest-road-edge}(\cdot)$ computes the distance between the agent and the nearest road edge.

B.3 Evaluation Metrics

We define the *Goal success* as the fraction of evaluated agents (over all test scenes) that come within 1 m of the ground-truth goal. The final and average displacement errors (FDE/ADE) are computed per agent over test scenes and then averaged. For each scene s , *collision* and *off-road* rates are the fractions of evaluated agents in s that collide with another agent or a road edge, respectively; the overall rates are the averages of these per-scene values.

We compare real vs. simulated rollouts via the Jensen–Shannon *distance* (JSD)

$$\sqrt{\frac{D_{\text{KL}}(p||m) + D_{\text{KL}}(q||m)}{2}}, \quad (\text{B.5})$$

using normalized histograms. Unlike prior work reporting the JSDivergence [125, 130], we use its square root, the JSDistance, for better readability. We compute JSD for linear speed, angular speed, acceleration, and nearest distance with the following bins: acceleration—21 uniform bins in $[-10, 10]$ (one per valid value); linear speed—200 bins in $[0, 30]$; angular speed—200 bins in $[-50, 50]$; nearest distance—200 bins in $[0, 40]$.

B.4 Individual JSD Results

In Table B.1, we report the per-feature JSD results for Table 6.4.

B.5 CtRL-Sim Training and Inference Details

Training. We train CtRL-Sim on randomly subsampled sequences of length $H \times N \times 3$ with $H=32$ and $N=24$. Actions are discretised into 20 (accel) $\times$ 50 (steer) bins, yielding 1000 action tokens. Each return-to-go component $G_t^{c,i}$ is quantised into 350 bins. Scenes are encoded in a global frame as in [27, 109]: we centre/rotate on a random agent, take the $L=200$ closest lane segments within 100 m (each subsampled to $P=100$ points), and up to $N=24$ nearest agents within 60 m as social context. The model uses hidden size $d=256$, $E=2$ Transformer encoder blocks and $D=4$ decoder blocks, with loss weight $\alpha=1/100$; supervision is applied only to moving agents. We apply *goal dropout* (zeroing 10% of goal embeddings) and set state/return/action embeddings to zero at missing timesteps. To maintain permutation equivariance[18, 109], we modify the causal mask so each agent at the current step can attend to *its own* action and return-to-go tokens, to *all* agents’ state tokens at the current step, and to *all* agents’ tokens in the past. Training uses AdamW with a linearly decaying LR from 5×10^{-4} over 200k steps, batch size 64. At inference we sample actions with temperature

Method	JSD ($\times 10^{-2}$)				
	Lin. Speed	Ang. Speed	Accel.	Nearest Dist.	Meta-JSD
Replay-Physics*	0.1	11.5	17.4	1.2	7.6
Actions-Only [27]	4.1 ± 0.7	16.8 ± 0.3	15.6 ± 0.5	5.1 ± 0.5	10.4 ± 0.3
Imitation Learning	1.0 ± 0.1	<u>13.4 ± 0.3</u>	16.9 ± 0.4	2.1 ± 0.2	8.3 ± 0.1
DT (Max Return) [140]	2.7 ± 0.1	<u>13.4 ± 0.2</u>	15.4 ± 0.5	2.2 ± 0.3	8.4 ± 0.1
CTG++ [132]	3.2 ± 0.9	11.9 ± 0.9	12.4 ± 0.4	2.2 ± 0.3	7.4 ± 0.2
CtRL-Sim (No State Prediction)	1.2 ± 0.1	13.7 ± 0.2	15.9 ± 0.7	<u>2.0 ± 0.2</u>	8.2 ± 0.2
CtRL-Sim (Base)	<u>1.1 ± 0.2</u>	13.8 ± 0.2	15.6 ± 0.5	<u>2.0 ± 0.3</u>	8.1 ± 0.2
CtRL-Sim (Positive Tilting)	1.4 ± 0.1	13.6 ± 0.2	<u>14.8 ± 0.5</u>	1.8 ± 0.2	<u>7.9 ± 0.1</u>
DT* (GT Initial Return)	1.1 ± 0.2	13.4 ± 0.2	16.8 ± 0.6	2.1 ± 0.2	8.4 ± 0.1
CtRL-Sim* (GT Initial Return)	1.1 ± 0.2	13.8 ± 0.3	15.3 ± 0.6	2.2 ± 0.2	8.1 ± 0.2

Table B.1 **Breakdown of Meta-JSD in Table 6.4.** For each metric, the best unprivileged method is **bolded** and second-best is underlined. * denotes a privileged method requiring the ground-truth future trajectory.

1.5. The CtRL-Sim architecture has 8.3M parameters and trains in ~ 20 hours on $4 \times A100$ GPUs.

Inference. CtRL-Sim supports arbitrary numbers of agents. When the number of CtRL-Sim -controlled agents exceeds $N=24$, we process per-timestep subsets of 24 agents: pick a controlled agent not yet processed, normalise the scene to it, add its 23 nearest neighbours, and repeat until all controlled agents are covered; if an agent appears in multiple subsets, we keep the model’s first prediction. We use context length $H=32$ (the most recent 32 steps) and center/rotate on the agent at the oldest context step. During the first 10 steps (1s) of rollout, states and actions are clamped to ground truth while the return-to-go is predicted at every step.

B.6 Baseline Details

In this section, we describe the design decision of each baseline employed in our work. We note that for all models below, we scaled them in order for all models to have approximately the same number of learnable parameters as CtRL-Sim’s architecture.

Actions Only The *actions-only* baseline is encoder-decoder architecture implemented in exactly the same way as CtRL-Sim with a few ablations. These include removal of states and returns from the decoder sequence, and no state rollout predictions. This model was inspired by [27] but differs in that the model also has access to the agents’ goals, and does not employ their k-disks action tokenization.

Imitation Learning The *imitation learning* baseline is also based on the CtRL-Sim multi-

agent behaviour simulation architecture but lacks factorized return information. It is a step better than the *actions-only* baseline since it considers the states in the decoder sequence, and this is corroborated by its improved performance on multi-agent simulation results of Table 6.4.

DT The *Decision Transformer* baseline is based on the seminal work [140]. We adopt an identical architecture to CtRL-Sim’s with some minor difference based on the algorithm to provide a fair comparison. The lack of a return prediction based on states, and instead returns are chosen at inference time based on domain knowledge. In DT, returns are the first token fed to the decoder, followed by states in order to predict actions. We show that this is suboptimal for controllability (results in Figure 5.4) and does not provide intuitive mechanism for selecting the return values to target.

CTG++ The *CTG++* baseline is our diffusion reimplementation of Zhong et al. [132], closely following their architecture with a few changes: (i) we diffuse jointly over *states and actions* (not actions plus a unicycle model), since Nocturne’s physics are not unicycle-governed and its forward model is non-differentiable; (ii) we condition on the current timestep and goal for parity with CtRL-Sim; (iii) we generate at 2 Hz as in CTG++, train with 100 diffusion steps but evaluate with 50—Table B.2 shows negligible differences; (iv) because we do not have information on the original network size, we explored different hidden dimension sizes of the transformer architecture (Table B.2); and (v) for future-relative encoding we use the last observed timestep which we found more stable. CTG++ uses ground truth at train time and constant velocity at test.

B.7 CAT Simulated Data Collection and CtRL-Sim Finetuning

The Waymo Open Motion dataset largely contains nominal driving scenes. To enhance control over the generation of safety-critical scenarios after initial pretraining, we finetune

Method	ADE (m) ↓	FDE (m) ↓	Goal Success Rate (%) ↑	JSD ($\times 10^{-2}$) ↓	Collision (%) ↓	Off Road (%) ↓	Per Scene Gen. Time (s) ↓
CTG++ [†]	1.72	3.97	41.7	7.7	6.4	17.4	44.0
CTG++ (128 hidden dim)	1.83	4.32	37.8	7.6	7.0	17.7	25.0
CTG++ (100 diffusion steps)	1.87	3.98	43.0	8.6	7.9	16.8	140.0

Table B.2 Ablations of CTG++ on Multi-agent simulation results over 1000 test scenes. We report the results across all metrics of different configurations of the CTG++ baseline [132]. [†] indicates the original model results, with 256 hidden dimension and 50 diffusion steps at inference time.

CtRL-Sim on a simulated dataset of safety-critical scenarios generated by CAT [224]. CAT is a state-of-the-art collision generation method that involves fixing the agent’s future trajectory to a trajectory predicted by a DenseTNT trajectory predictor. In its essence, CAT searches for a trajectory that has high likelihood of colliding with the log-replay future trajectory of an ego vehicle, while having high probability under the behaviour prior (DenseTNT). For more details, we refer readers to [224]. A notable limitation of CAT is its lack of reactivity to the ego since the agent’s trajectory is fixed at the beginning of the simulation. Moreover, unlike CtRL-Sim, CAT cannot control the degree to which the agent is adversarial.

To collect the simulated safety-critical dataset, we run CAT on a subset of the interactive validation split of the Waymo Open Motion Dataset, which involves two interacting agents. Following CAT’s methodology, we select one of the two interacting agents to be the ego (whose trajectory is fixed to the log-replay trajectory) and the other interacting agent to be the CAT adversary. In total, we collect 3577 CAT scenarios for finetuning, of which around 60% contain ego-adversary collisions. To encourage CtRL-Sim to learn how to generate safety-critical scenarios without forgetting how to generate good driving behaviour, we adopt a continual pre-training strategy for finetuning [277] where we randomly sample 3577 real training scenarios from the offline RL dataset in each training epoch, or a 50% replay ratio. We rewarm the learning rate to the maximum learning rate of $5e-4$ over 500 steps and follow a linear decay learning rate schedule to 0 over 20 epochs. The finetuned CtRL-Sim model should be more capable of generating long-tail scenarios as it is more exposed to such scenarios during finetuning. Finetuning takes roughly 30 minutes on 1 NVIDIA A100-Large GPU.

B.8 Additional Qualitative Results

Figure B.1 presents additional qualitative results illustrating the effect of *positive* exponential tilting on each reward component. In the left panels, CtRL-Sim without tilting yields a vehicle–vehicle collision at a left turn; with positive vehicle–vehicle tilting, the CtRL-Sim-controlled agent shifts toward the right side of the lane and avoids contact. In the middle panels, no tilting leads to a vehicle–edge collision as a bus pulls to the curb; with positive vehicle–edge tilting, the agent maintains a safer offset. In the right panels, the agent reaches the goal without tilting; with positive goal tilting, it arrives much faster and nearly avoids a conflict with a turning vehicle. Figure B.2 shows two further adversarial-collision cases generated with *negative* vehicle–vehicle tilting.

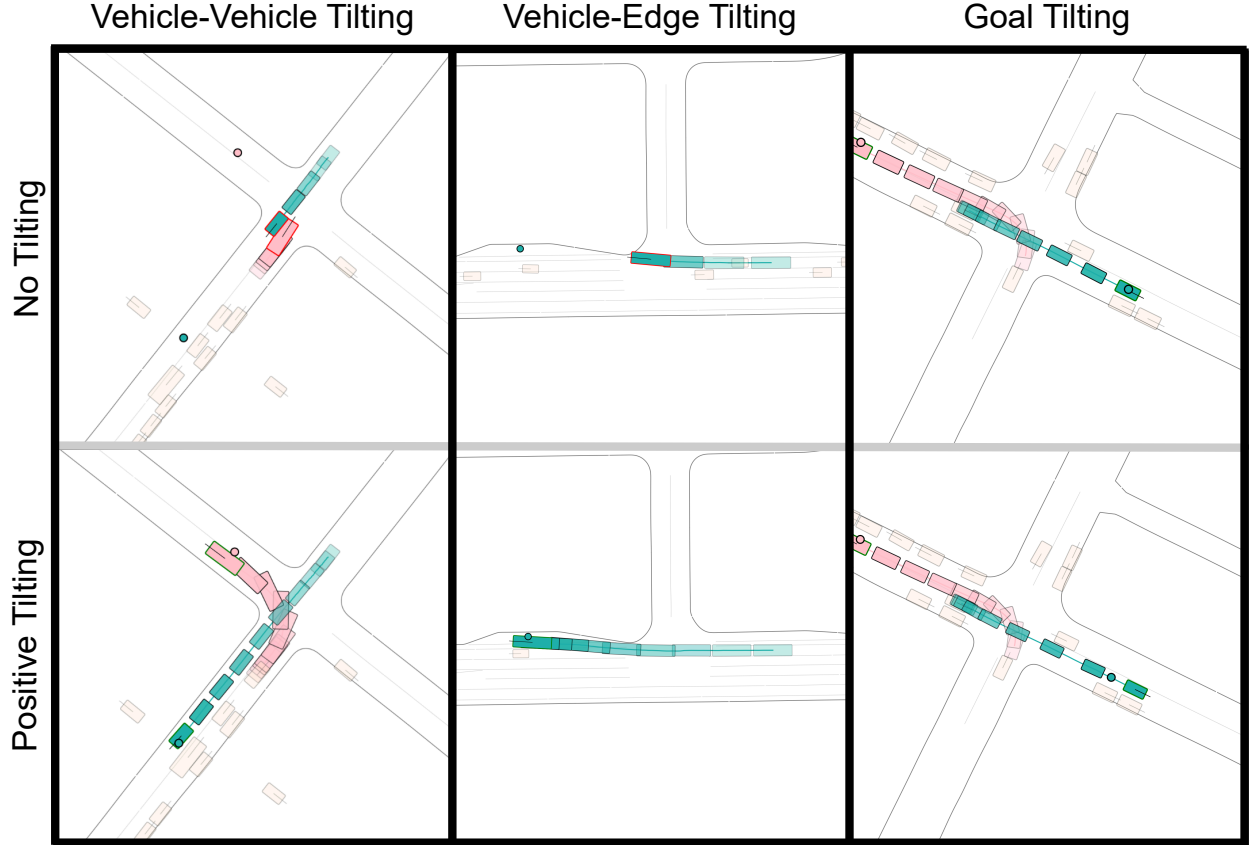


Figure B.1 **Qualitative results of the effects of positive tilting [26]**. We show the evolution of three traffic scenes with the top panels applying no exponential tilting to the CtRL-Sim-controlled agent (shown in teal) and the bottom panels applying positive tilting to the same CtRL-Sim-controlled agent. Bounding boxes outlined in red contain a traffic violation. All other agents are set to log-replay through physics, with the agent interacting with the CtRL-Sim-controlled agent denoted in pink. Goals are denoted by small circles.

B.9 Multi-Agent Simulation Results with Higher Temperature Sampling

In Table B.3, we report results from the same experiments as Table 5.1 except with an action sampling temperature of 1.5.

B.10 Fine-tuning CtRL-Sim on CtRL-Sim Scenarios

Instead of fine-tuning on CAT scenarios, we fine-tune CtRL-Sim on adversarial cases it generates itself. We construct a simulated dataset of scenarios by rolling out a single CtRL-Sim-controlled agent with negative tilts while all other agents replay logs through physics, and we

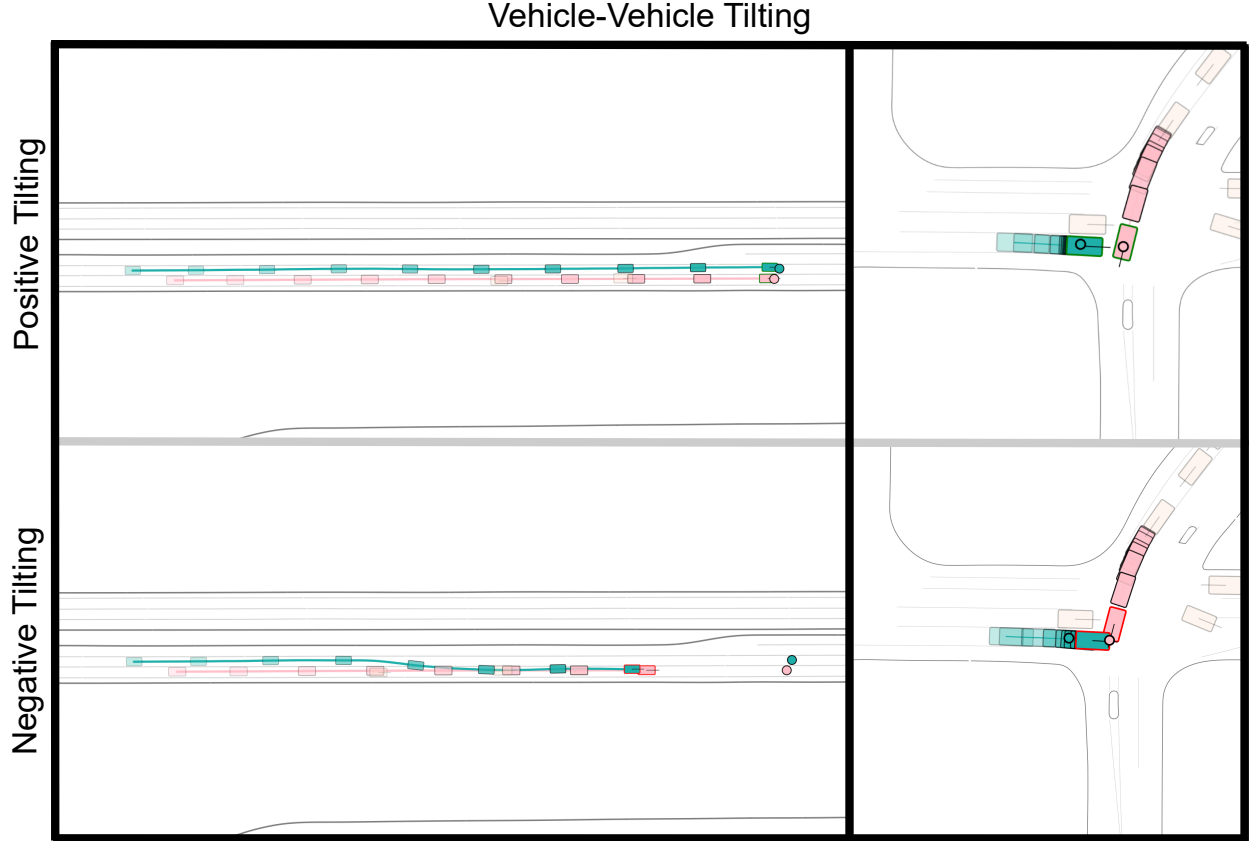


Figure B.2 **Qualitative results of vehicle-vehicle tilting** [26]. We show the evolution of two traffic scenes with the top panels applying positive exponential tilting to the CtRL-Sim-controlled agent (shown in teal) and the bottom panels applying negative tilting to the same CtRL-Sim-controlled agent. Bounding boxes outlined in red contain a traffic violation. All other agents are set to log-replay through physics, with the agent interacting with the CtRL-Sim-controlled agent denoted in pink. Goals are denoted by small circles.

keep a scene only if it yields a vehicle-vehicle or vehicle-road-edge collision. For tilting, for the vehicle-vehicle cases we sample $\kappa_{\text{veh-veh}}, \kappa_{\text{goal}} \sim \mathcal{U}(-25, 0)$, while for the vehicle-road-edge cases we sample $\kappa_{\text{veh-edge}}, \kappa_{\text{goal}} \sim \mathcal{U}(-25, 0)$. Negatively tilting the goal relaxes goal-seeking, making violations easier to elicit. We collect 5,000 scenarios of each type to form a safety-critical dataset. Fine-tuning follows Appendix B.7 with a larger replay ratio: each epoch mixes in 90,000 real scenarios from the offline RL set (90% replay), which we found helped avoid forgetting how to generate good driving behaviour. Figure B.3 shows controllability comparable to the CAT-finetuned CtRL-Sim model.

Method	FDE (m)	Reconstruction		Distributional Realism Meta JSD($\times 10^{-2}$)	Common Sense	
		ADE (m)	Goal Suc. Rate (%)		Collision (%)	Off Road (%)
Replay-Physics*	0.97	0.47	87.3	7.6	2.8	10.7
Actions-Only [27]	11.70 ± 1.12	4.78 ± 0.42	34.4 ± 1.3	14.3 ± 0.3	22.8 ± 0.7	29.7 ± 1.7
Imitation Learning	2.42 ± 0.17	1.47 ± 0.07	73.8 ± 1.2	12.3 ± 0.5	7.3 ± 0.6	13.1 ± 0.4
DT (Max Return) [140]	3.25 ± 0.17	1.67 ± 0.05	60.5 ± 1.2	12.3 ± 0.4	6.1 ± 0.7	11.6 ± 0.3
CtRL-Sim (No State Prediction)	2.57 ± 0.16	1.52 ± 0.07	66.2 ± 1.0	12.3 ± 0.3	7.6 ± 0.7	13.1 ± 0.3
CtRL-Sim (Base)	2.49 ± 0.10	1.50 ± 0.04	67.9 ± 1.2	12.2 ± 0.2	7.6 ± 0.3	13.1 ± 0.5
CtRL-Sim (Positive Tilting)	2.38 ± 0.08	1.44 ± 0.03	67.2 ± 1.0	12.1 ± 0.1	6.7 ± 0.4	12.3 ± 0.3
DT* (GT Initial Return)	1.94 ± 0.07	1.28 ± 0.02	73.7 ± 1.5	12.2 ± 0.3	6.6 ± 0.4	12.6 ± 0.4
CtRL-Sim* (GT Initial Return)	1.97 ± 0.08	1.30 ± 0.03	71.1 ± 0.9	12.2 ± 0.1	7.2 ± 0.5	13.1 ± 0.3

Table B.3 **Multi-agent simulation results over 1000 test scenes with action temperature = 1.5 over 3 seeds.** This table presents the results from the same experiments as Table 6.4, but with an action sampling temperature of 1.5 instead of 1.0. This allows for a comparison of the impact of the temperature hyperparameter. Overall, an action sampling temperature of 1.0 yields better results.

B.11 Dynamic Exponential Tilting

We study *dynamic exponential tilting*, where the tilt changes at a timestep, focusing on *late tilting* to test its effectiveness at later timesteps. Concretely, we apply a -10 tilt to the CtRL-Sim FT model only in the final half of the rollout (last 4s), evaluate each reward component independently on 1,000 scenes across 5 seeds (Fig. 5.4), and report mean $\pm$ std in Table B.4. Compared to applying negative tilting at all timesteps, late tilting yields higher goal success and lower collision/off-road rates; compared to no tilting, it yields lower goal success and higher violation rates, showing tilting remains effective even when applied late. Notably, late tilting performs similarly to full tilting on the goal return-to-go, likely because goal avoidance can be achieved within the last 4s, whereas provoking vehicle-vehicle or road-edge collisions late is harder without early planning.

Tilting	Goal Success Rate (%)	Tilting	Collision Rate (%)	Tilting	Offroad Rate (%)
0	69.0 ± 0.7	0	14.5 ± 1.5	0	15.2 ± 0.9
-10 (Late)	64.1 ± 1.3	-10 (Late)	18.2 ± 2.3	-10 (Late)	17.4 ± 1.0
10	63.1 ± 2.6	10	28.9 ± 1.8	10	21.6 ± 0.6
(a) Goal Tilting		(b) Vehicle-Vehicle Tilting		(c) Vehicle-Edge Tilting	

Table B.4 **CtRL-Sim-FT’s titling Results.** We report the results of the CtRL-Sim FT model over 1000 test scenes applying (a) Goal Tilting, (b) Vehicle-Vehicle Tilting, and (c) Vehicle-Edge Tilting. We either apply no tilting, -10 tilting, or no tilting for the first half of the rollout and -10 tilting for the second half of the rollout (*i.e.*, Late). We report the mean $\pm$ std over 5 seeds.

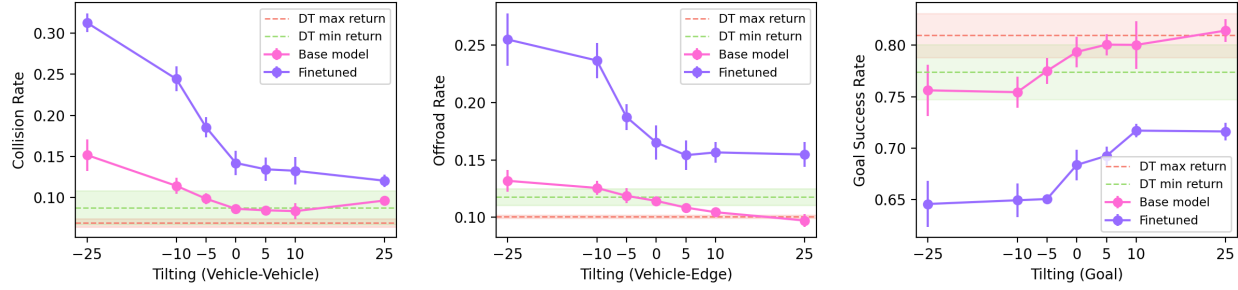


Figure B.3 **Effects of exponential tilting [26]**. Comparison of CtRL-Sim base model (magenta) and a CtRL-Sim model fine-tuned on adversarial CtRL-Sim scenarios (purple). As opposed to Figure 5.4, this fine-tuned model does not involve using CAT to select the adversarial scenarios. Rewards range from -25 to 25 for vehicle-vehicle collision (**left**), vehicle-edge collision (**middle**), and goal reaching (**right**). Results show smooth controllability, with fine-tuning enhancing this effect. Mean and std are reported over 5 seeds.

B.12 Failure Cases

Figure B.4 illustrates two CtRL-Sim failure modes. On the left, a CtRL-Sim -controlled agent (green) with zero tilting drifts off-road, which is likely caused by imitation drift, due to limited training exposure to recovery near road edges. On the right, using strong negative vehicle-vehicle tilting ($\kappa_{\text{veh-veh}} = -50$) produces erratic, implausible behaviour by pushing the model out of distribution. This can be mitigated by moderating the tilt; empirically, values in $[-25, 25]$ yield more plausible outcomes.

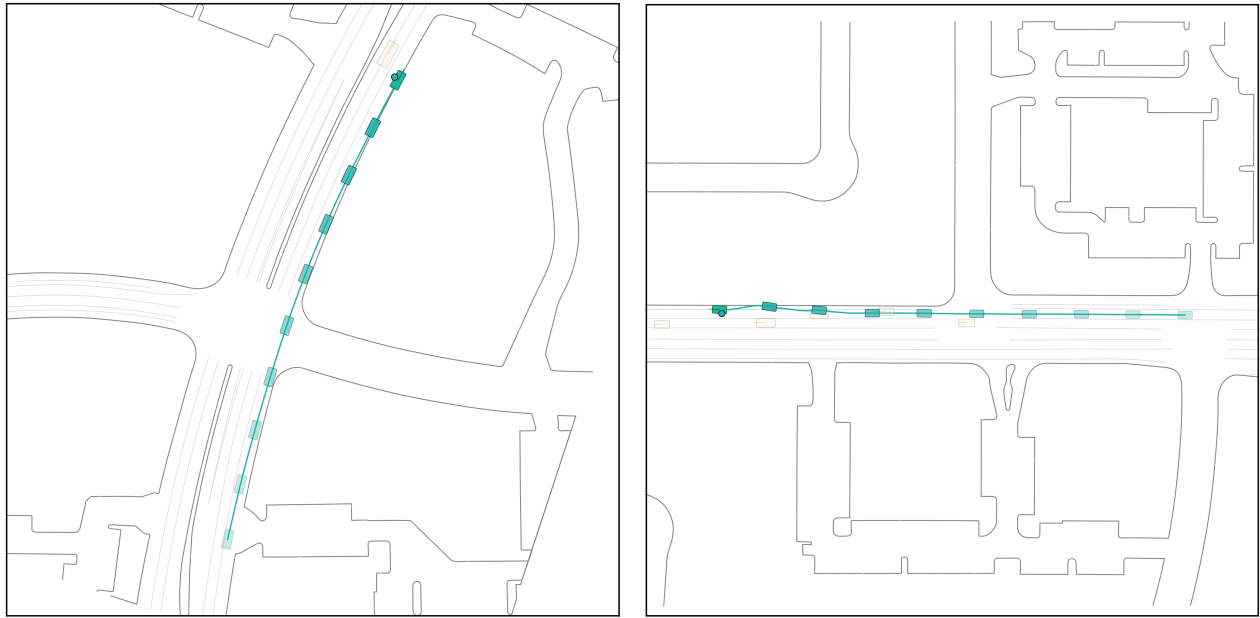


Figure B.4 **Failure cases** [26]. Visualization of two failure cases of CtRL-Sim. The agent controlled by CtRL-Sim is depicted in teal. On the left, the agent drifts off the road, likely due to imitation drift. On the right, the agent's behaviour is erratic and implausible due to extreme tilting values (-50).

APPENDIX C VECTORIZED LATENT DIFFUSION & SCENARIO DREAMER

C.1 Additional Results

C.1.1 Behaviour Simulation

Method	Control?	JSD ($\times 10^{-2}$) $\downarrow$				Agent Collision	Planner Collision
		Lin. Spd.	Ang. Spd.	Acc.	Near. Dist.	(%) $\downarrow$	(%) $\downarrow$
IDM [32]	$\times$	9.2	0.4	19.8	1.6	7.2	5.8
Trajeglish [†] [240]	$\times$	19.5	0.3	19.7	4.0	6.4	7.0
CtRL-Sim (Positive Tilting)	$\checkmark$	4.1	0.1	20.1	1.3	6.2	4.9
CtRL-Sim (Negative Tilting)	$\checkmark$	4.2	0.2	26.1	1.5	10.9	11.9

Table C.1 **Simulation Methods Comparison.** Comparison of different methods for behaviour simulation over 1000 Waymo test scenes. [†] indicates reimplementatation. (© 2025 IEEE)

Since we use a modified CtRL-Sim model for Scenario Dreamer, we provide additional simulation experiments in this section. We detail its architectural changes in appendix C.3.2.

Table C.1 reports behaviour-simulation results on held-out WOMD test scenes. We compare CtRL-Sim to IDM [32] (used in SLEDGE [93]) and Trajeglish [240]. All methods use the Scenario Dreamer lane graph (lanes resampled to 50 points) and simulate within an 80 m $\times$ 80 m FOV around the ego; IDM assigns each agent a random lane-graph route, while the ego follows an IDM planner on the route closest to the logged trajectory. We report JSDs (linear/angular speed, acceleration, nearest distance) and collision rates—*Agent Collision* and *Planner Collision*. To assess controllability we also include exponentially tilted CtRL-Sim variants with $\kappa = +10$ and $\kappa = -50$.

CtRL-Sim performs competitively against both baselines across JSD and collision metrics. The positively tilted model attains the best JSDs (except acceleration) and the lowest planner-collision rate (4.9%), whereas IDM and Trajeglish show higher planner collisions, indicating weaker coordination with the IDM planner. Negative tilting increases planner collisions by 7 pp with only modest JSD degradation, demonstrating controllability, while remaining realistic evidenced by only a modest increase in JSD metrics. Notably, this CtRL-Sim variant is *not* fine-tuned on adversarial data, unlike the variant in Chapter 5.

C.1.2 Qualitative Results

Figure C.1 shows Scenario Dreamer L in *lane-conditioned object generation* and *initial scene generation* (trained on nuPlan and Waymo). Random initial scenes in Figs. C.3 and C.4 compare our model to a retrained SLEDGE DiT-XL [93]: despite $\sim 100\text{M}$ fewer parameters, $4\times$ lower inference latency, and $\sim 4\times$ fewer GPU hours, Scenario Dreamer L yields higher-quality scenes. When contrasting rasterized ground truth processed by DriveSceneGen [92] with our samples, the vectorization pipeline often produces lane-graph errors, whereas Scenario Dreamer L generates more realistic lane geometries. This demonstrates the effectiveness of the proposed vectorized latent diffusion modelling approach.

Figure C.2 highlights inpainting near complex lane borders with Scenario Dreamer L model, where we are able to inpaint at the border of complex lane geometries, such as intersections. The project webpage visualizes the diffusion chain across generation modes. Figure C.5 demonstrates diversity under the same initial conditions: (top) multiple inpaintings of the same partial scene; (bottom) full scenes conditioned on 24 lanes and 8 agents—both. We see that both approaches yield plausible variations of the initial scene. Figure C.6 shows qualitative closed-loop CtRL-Sim rollouts within Scenario Dreamer environments.

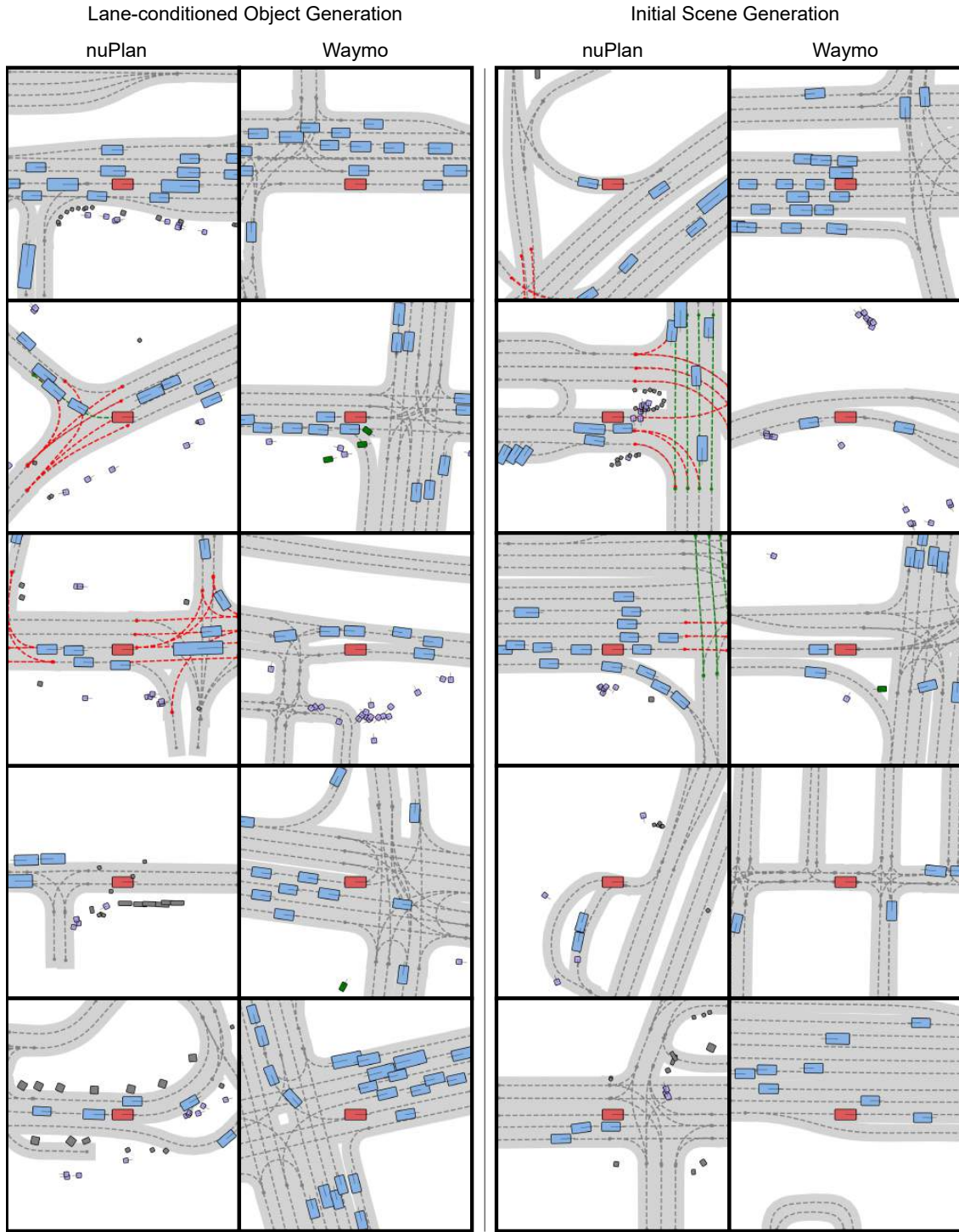


Figure C.1 **Lane-conditioned object generation and initial scene generation qualitative results** [44]. We visualize samples from the Scenario Dreamer (L) model in lane-conditioned object generation mode (columns 1 and 2) and full scene generation mode (columns 3 and 4) on both the nuPlan and Waymo datasets. (© 2025 IEEE)

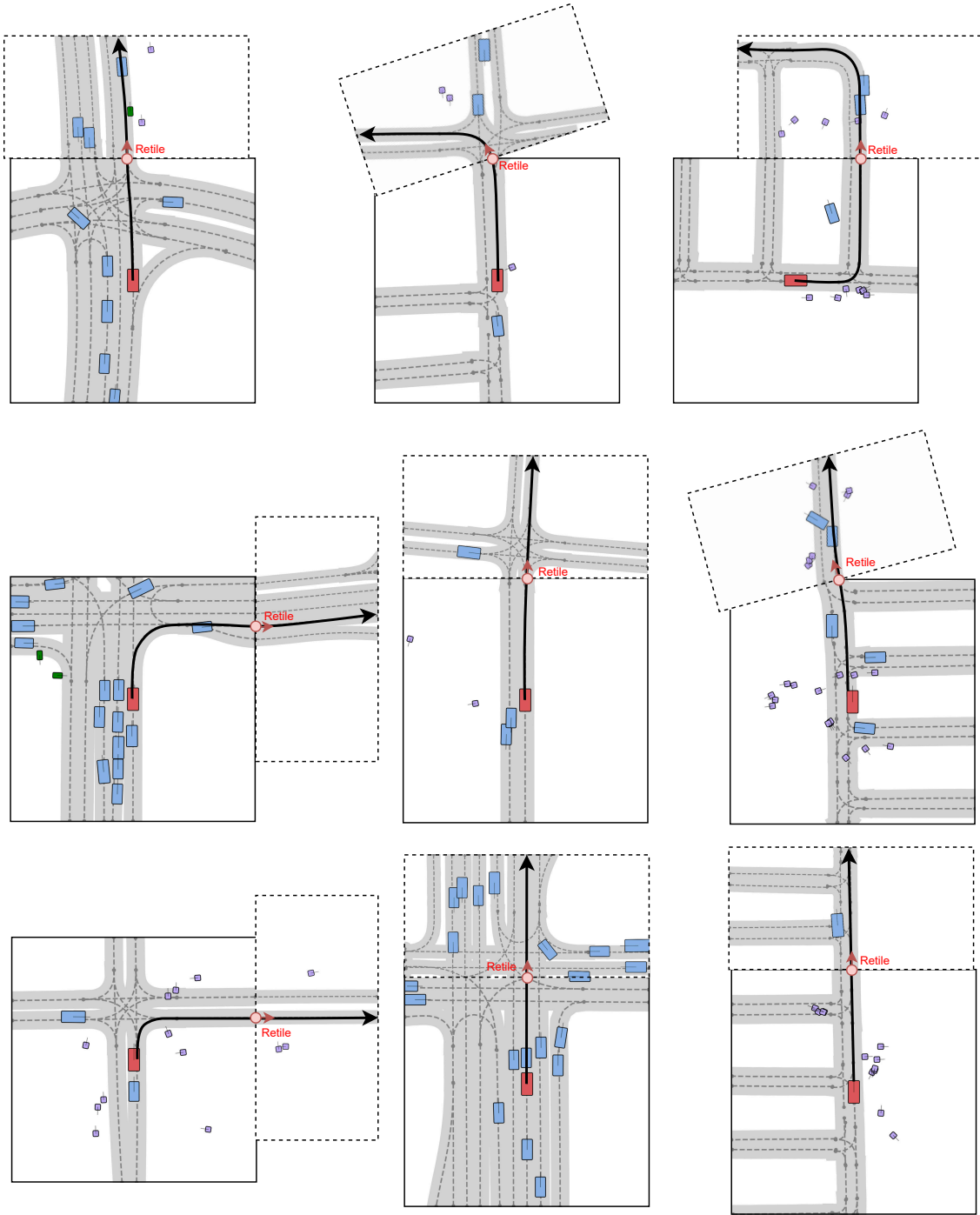


Figure C.2 **Scenario Dreamer inpainting** [44]. We visualize the inpainting capabilities of Scenario Dreamer trained on the Waymo dataset. The initial tile is outlined in a solid line, and the new inpainted tile is outlined in a dashed line. The ego route is visualized as a solid black line. (© 2025 IEEE)

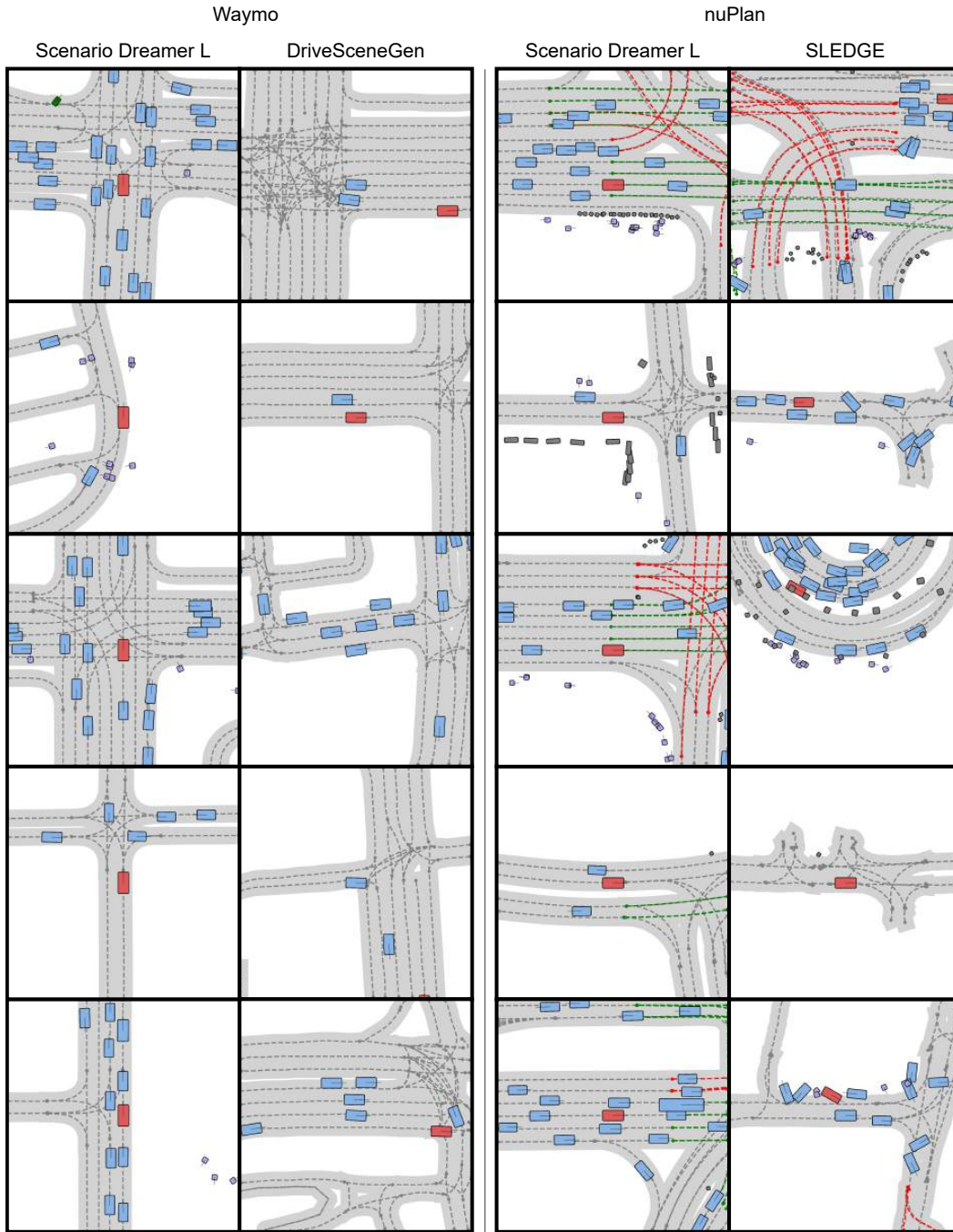


Figure C.3 **Qualitative results comparison across methods (set 1)** [44]. We visualize samples from the Scenario Dreamer (L) model trained on the Waymo dataset and the privileged DriveSceneGen model (columns 1 and 2) along with samples from Scenario Dreamer (L) and SLEDGE DiT-XL both trained on nuPlan (columns 3 and 4). (© 2025 IEEE)

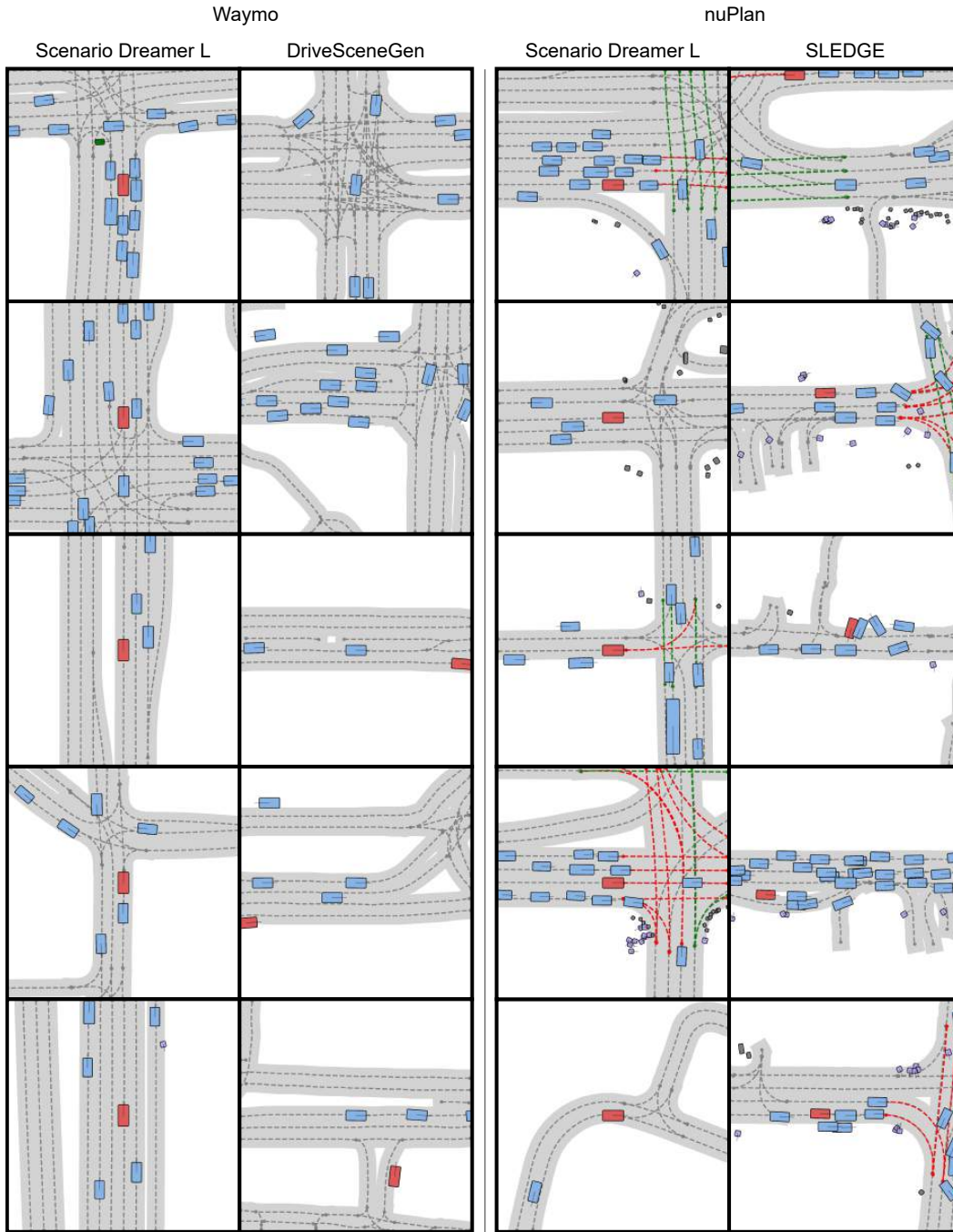


Figure C.4 **Qualitative results comparison across methods (set 2)** [44]. We visualize samples from the Scenario Dreamer (L) model trained on the Waymo dataset and the privileged DriveSceneGen model (columns 1 and 2) along with samples from Scenario Dreamer (L) and SLEDGE DiT-XL both trained on nuPlan (columns 3 and 4). (© 2025 IEEE)

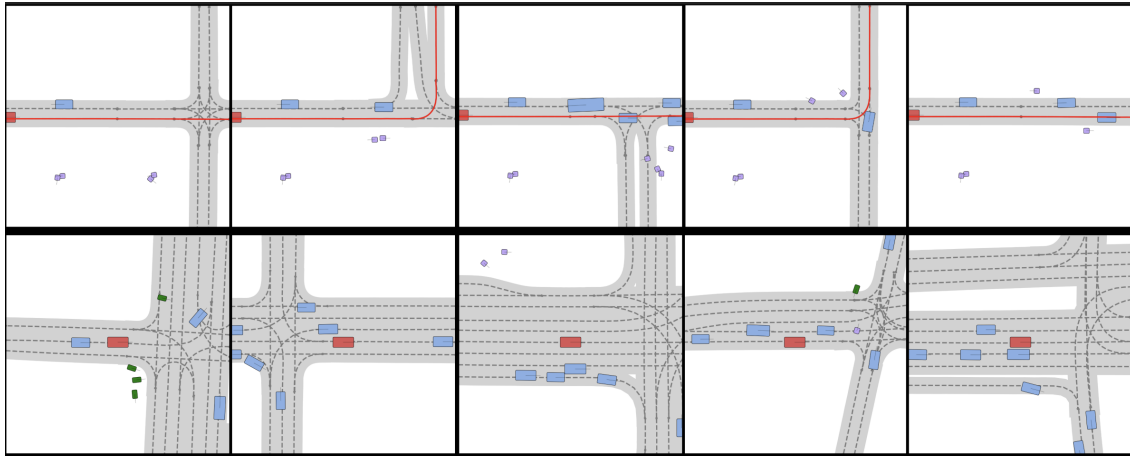


Figure C.5 **Scene Diversity** [44]: (Top) Right-half of the scene inpainted from the same left-half of the scene. (Bottom) Random samples with the same initial conditions: 8 agents and 24 lanes. (© 2025 IEEE)

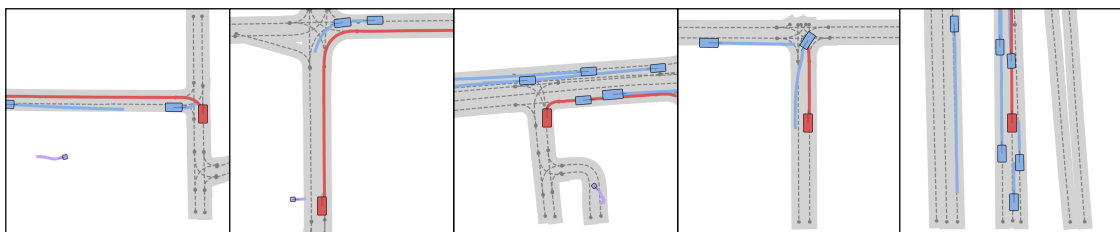


Figure C.6 **Behaviour Simulation Examples** [44]: CtRL-Sim rollouts (blue) with IDM ego planner (red) simulated from SD initial scenes. (© 2025 IEEE)

C.2 Simulation Framework Details

C.2.1 Simulator Properties

Scenario Dreamer enables evaluation of AV planners within its generative environments over arbitrarily long horizons, for example, along a 500 m route. We first sample an initial scene $\mathcal{I}_F$ and select a route represented by lane centerlines resampled at 1 m intervals, obtained via a depth-first traversal of the lane graph from the origin. We then apply an SE(2) transform to align (renormalise) $\mathcal{I}_F$ to the end of the current route segment, and the vectorized latent diffusion model inpaints the next segment $\mathcal{I}_{F_P}$ conditioned on the existing map $\mathcal{I}_{F_N}$. This cycle of route extension and inpainting repeats until the desired route length is achieved.

Scenario Dreamer, trained on the Waymo dataset, provides the generative environments for AV-planner evaluation. Maps use a centerline representation with 50 points per lane segment plus connectivity. The simulator supports vehicles, pedestrians, and bicyclists. Non-planner agents are controlled by the trained CtRL-Sim model acting in $(\Delta x, \Delta y, \Delta \theta)$ with actions drawn from the k -disks vocabulary (Fig. C.9). The AV planner outputs steering and acceleration, which are integrated using a forward bicycle model [87] to update the state. At each timestep the planner observes a 64 m radius context, while CtRL-Sim simulates agents within a slightly larger 80m×80m field of view (FOV) around the AV’s position. This restricted FOV minimizes simulation latency, while agents outside the radius remain in a constant state until they enter the FOV; if an agent exits the FOV, they are not allowed to re-enter. This limited FOV allows us to achieve efficiency using a lightweight 8M parameter behaviour model, which is at least 5× smaller than several state-of-the-art behaviour models [119, 240, 278]. The planner’s objective is to complete the route within a time limit without collisions and with path deviation < 2.5 m; a run fails on any collision, excessive deviation, or timeout.

C.2.2 Adversarial Simulation Environments

Scenario Dreamer supports the generation of adversarial simulation environments by leveraging the controllability of CtRL-Sim through exponential tilting. For adversarial episodes, one agent from within the simulation radius is sampled from a negatively tilted return distribution. Recall that this is rewarding collision behaviours with the ego vehicle. Table C.1 shows that we can indeed produce significantly more collisions through tilting.

C.2.3 Training Scenario Dreamer-compatible RL Agents in GPU Drive

We train PPO [62] planning agents on 100 simulation-compatible Waymo scenarios for 100 million steps using the GPU-accelerated GPU Drive simulator [87]. Training requires approximately 24 hours on a single A100-L GPU. To ensure compatibility with Scenario Dreamer environments, we make four key modifications to the GPU Drive simulator:

1. **Route-conditioning:** We replace goal-conditioning with route-conditioning, as Scenario Dreamer provides a route for the planner to follow. The route is defined by the sequence of lane centerlines nearest to the logged ground-truth trajectory of the ego.
2. **Map simplification:** We remove road edges and crosswalks from the map, retaining only lane centerlines that are preprocessed in the same way as in Scenario Dreamer.
3. **Route deviation penalty:** To encourage on-road driving in the absence of road edges, we penalize the planner for lateral deviations exceeding $2.5 + \text{width} = 4.83\text{m}$ from the lane centerline.
4. **Episode termination:** Episodes end only when the ego either reaches the end of the prescribed route (i.e., the goal) or when the episode times out after 9 seconds.

The reward function consists of three components: a collision penalty with weight 0.1, a route-deviation penalty with weight 0.05, and a goal-achievement reward with weight 1.

C.3 Model Details

C.3.1 Vectorized Latent Diffusion Model

Autoencoder

Encoder The autoencoder’s encoder $\mathcal{E}_\phi$ takes as input a set of lanes $\{\mathbf{l}_i\}_{i=1}^{N_l}$, a set of objects $\{\mathbf{o}_i\}_{i=1}^{N_o}$, and a lane-connectivity tensor $\mathbf{A} \in \{0, 1\}^{N_l \times N_l \times 4}$. Separate 2-layer MLPs, f_l and f_o , embed lane and object vectors, respectively; lane/object type are concatenated to the inputs before the MLPs.

$$\begin{aligned}\mathbf{l}_i^0 &= f_l([\mathbf{l}_i, c_i^l]), \\ \mathbf{o}_i^0 &= f_o([\mathbf{o}_i, c_i^o]),\end{aligned}$$

where c_i^l is the lane type of lane i and c_i^o is the object type of object i . A 2-layer MLP f_a additionally processes the lane connectivity type $a_{ij} := \mathbf{A}[i, j]$ between all (directed) edges

(i, j) connecting lane i to lane j

$$\mathbf{a}_{ij}^0 = f_a(a_{ij}).$$

The lane, object, and lane connectivity embeddings are then processed by a sequence of N_E factorized attention blocks (FABs), where each FAB consists of sequential lane-to-lane (L2L), lane-to-object (L2O), and object-to-object (O2O) attention layers. First, we define $\mathbb{L}^k := \{\mathbf{l}_i^k\}_{i=1}^{N_l}$, $\mathbb{O}^k := \{\mathbf{o}_i^k\}_{i=1}^{N_o}$, and $\mathbb{A}^k := \{\mathbf{a}_{ij}^k\}_{(i,j) \in [N_l] \times [N_l]}$ as the set of embeddings for the lanes, objects, and lane connectivities following the output of the k 'th FAB. The k 'th FAB updates the k 'th layer lane, object, and lane connectivity embeddings as

$$(\mathbb{L}^{k+1}, \mathbb{O}^{k+1}, \mathbb{A}^{k+1}) = \text{FAB}_k(\mathbb{L}^k, \mathbb{O}^k, \mathbb{A}^k),$$

where FAB_k is decomposed as follows

$$\begin{aligned} \mathbb{L}^{k+1} &= \text{L2L}_k(\mathbb{L}^k, \mathbb{A}^k), \\ \mathbb{O}^{k+1} &= \text{L2O}_k(\text{proj}^k(\mathbb{L}^{k+1}), \mathbb{O}^k), \\ \mathbb{O}^{k+1} &= \text{O2O}_k(\mathbb{O}^{k+1}), \\ \mathbb{A}^{k+1} &= \text{EdgeUpdate}_k(\mathbb{L}^{k+1}, \mathbb{A}^k) \end{aligned}$$

where L2L is a Transformer encoder block [236] whose multi-head self-attention incorporates the edge features $\mathbb{A}^k$ into the keys and values, following [279]. L2O is a Transformer decoder block in which lane embeddings are first projected to the object hidden dimension via a linear layer proj_k , and O2O is a Transformer encoder block. Because we operate on sets and require permutation equivariance, L2L, L2O, and O2O omit positional encodings. Finally, EdgeUpdate_k updates the lane-connectivity embeddings $\mathbf{a}_{ij}^k$ as

$$\mathbf{a}_{ij}^{k+1} = f_k^2([\mathbf{a}_{ij}^k, f_k^1([\mathbf{l}_i^{k+1}, \mathbf{l}_j^{k+1}])]),$$

where $[\cdot, \cdot]$ denotes concatenation along the feature dimension, and f_k^1, f_k^2 are each 2-layer MLPs. For the partitioned scenes, to predict the number of lanes in $\mathcal{I}_{F_P}$, we additionally define a learnable query vector $\mathbf{q}$ and each FAB_k in the encoder is augmented with a lane-to-query (L2Q) attention layer following the L2L layer

$$\mathbb{Q}^{k+1} = \text{L2Q}_k(\mathbb{L}_{F_N}^{k+1}, \mathbb{Q}^k),$$

where $\mathbb{Q}^0 = \{\mathbf{q}\}$ and $\mathbb{L}_{F_N}^{k+1}$ denotes the embeddings of the lanes in $\mathcal{I}_{F_N}$ of a partitioned scene.

After N_E FABs, the lane and object embeddings are projected to Gaussian mean/variance parameters with latent dimensions K_l and K_o , respectively, as in a standard VAE. The transformed query vector $\mathbf{q}^{N_E}$ is then passed through a 3-layer MLP f_{num} to predict the number of lanes in $\mathcal{I}_{F_P}$.

$$\begin{aligned}\boldsymbol{\mu}_{\mathcal{L}}^i &= f_{\text{mean}}^{\mathcal{L}}(\mathbf{l}_i^{N_E}), \\ \boldsymbol{\sigma}_{\mathcal{L}}^i &= f_{\text{std}}^{\mathcal{L}}(\mathbf{l}_i^{N_E}), \\ \boldsymbol{\mu}_{\mathcal{O}}^i &= f_{\text{mean}}^{\mathcal{O}}(\mathbf{o}_i^{N_E}), \\ \boldsymbol{\sigma}_{\mathcal{O}}^i &= f_{\text{std}}^{\mathcal{O}}(\mathbf{o}_i^{N_E}), \\ \hat{N}_l^{F_P} &= f_{\text{num}}(\mathbf{q}^{N_E})\end{aligned}$$

where $f_{\text{mean}}^{\mathcal{L}}, f_{\text{std}}^{\mathcal{L}}, f_{\text{mean}}^{\mathcal{O}}, f_{\text{std}}^{\mathcal{O}}$ are linear layers.

Decoder Given sampled latents $\{\{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o}, \{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l}\} \sim \mathcal{E}_{\phi}$ from the encoder, separate 2-layer MLPs f_{h_l} and f_{h_o} process the lane and object latents, respectively

$$\begin{aligned}\mathbf{l}_i^0 &= f_{h_l}(\mathbf{h}_i^{\mathcal{L}}), \\ \mathbf{o}_i^0 &= f_{h_o}(\mathbf{h}_i^{\mathcal{O}}).\end{aligned}$$

We additionally derive lane connectivity features $\mathbf{a}_{ij}^0$ by processing the incident lane embeddings $\mathbf{l}_i^0, \mathbf{l}_j^0$ through a 2-layer MLP f_{h_a} for all i, j

$$\mathbf{a}_{ij}^0 = f_{h_a}([\mathbf{l}_i^0, \mathbf{l}_j^0])$$

Similarly to the encoder, we then process the lane, object, and lane connectivity embeddings with a stack of N_D FABs. The resulting embeddings are used to predict the lane positions, lane type, object features, object type, and lane graph connectivity, for all i, j

$$\begin{aligned}\hat{\mathbf{l}}_i &= f_{\text{lane}}(\mathbf{l}_i^{N_D}), \\ \hat{\mathcal{C}}_i^l &= f_{\text{lane-type}}(\mathbf{l}_i^{N_D}), \\ \hat{\mathbf{o}}_i &= f_{\text{object}}(\mathbf{o}_i^{N_D}), \\ \hat{\mathcal{C}}_i^o &= f_{\text{object-type}}(\mathbf{o}_i^{N_D}), \\ \hat{a}_{ij} &= f_{\text{connectivity}}(\mathbf{a}_{ij}^{N_D}),\end{aligned}$$

where $f_{\text{lane}}, f_{\text{object}}$ are 4-layer MLPs and $f_{\text{lane-type}}, f_{\text{object-type}}, f_{\text{connectivity}}$ are 3-layer MLPs.

The autoencoder loss function is defined by

$$\begin{aligned}
L_{\text{ae}} = & \underbrace{\lambda_{\text{lane}} \frac{1}{N_l} \sum_{i=1}^{N_l} \left(\ell_2(\mathbf{l}_i, \hat{\mathbf{l}}_i) + \text{ce}(\hat{c}_i^l, c_i^l) \right) + \frac{1}{N_o} \sum_{i=1}^{N_o} \left(\ell_2(\mathbf{o}_i, \hat{\mathbf{o}}_i) + \text{ce}(\hat{c}_i^o, c_i^o) \right) + \lambda_{\text{conn}} \frac{1}{N_l^2} \sum_{i=1}^{N_l} \sum_{j=1}^{N_l} \text{ce}(\hat{a}_{ij}, a_{ij})}_{\text{reconstruction loss}} \\
& - \underbrace{\frac{\beta}{2} \left(\frac{1}{N_l} \sum_{i=1}^{N_l} \left[1 + \log(\boldsymbol{\sigma}_i^{\mathcal{L}})^2 - (\boldsymbol{\mu}_i^{\mathcal{L}})^2 - (\boldsymbol{\sigma}_i^{\mathcal{L}})^2 \right] + \frac{1}{N_o} \sum_{i=1}^{N_o} \left[1 + \log(\boldsymbol{\sigma}_i^{\mathcal{O}})^2 - (\boldsymbol{\mu}_i^{\mathcal{O}})^2 - (\boldsymbol{\sigma}_i^{\mathcal{O}})^2 \right] \right)}_{\text{KL loss}} + \lambda_{\text{num}} \text{ce}(\hat{N}_l^{FP}, N_l)
\end{aligned}$$

where λ_{lane} , λ_{conn} , λ_{num} are coefficients to scale the respective lanes, $\text{ce}(\cdot, \cdot)$ denotes the cross entropy loss, and β scales the KL loss. For non-partitioned scenes, $\lambda_{\text{num}} = 0$.

Latent Diffusion Model

Since we already covered the (latent) diffusion model setup in Chapter 2, we omit it from this appendix. The only difference is that instead of modelling x_0 , the latent diffusion model models the joint distribution over lane and object latents: $p_{\theta}(\{\mathbf{h}_i^{\mathcal{L}}\}_{i=1}^{N_l}, \{\mathbf{h}_i^{\mathcal{O}}\}_{i=1}^{N_o} | N_o, N_l)$. We let $\mathbf{H}_0$ denote the matrix of stacked lane and object latents. Then, the diffusion model models the distribution: $p_{\theta}(\mathbf{H}_0 | N_o, N_l)$.

Architecture We decompose $\mathbf{H}_t$ into lane latents $\mathbf{H}_t^{\mathcal{L}}$ and object latents $\mathbf{H}_t^{\mathcal{O}}$. The latent diffusion model ϵ_{θ} takes as input N_l noised lane latents and N_o noised object latents $(\mathbf{H}_t^{\mathcal{L}}, \mathbf{H}_t^{\mathcal{O}}) := \{\{\mathbf{h}_{i,t}^{\mathcal{L}}\}_{i=1}^{N_l}, \{\mathbf{h}_{i,t}^{\mathcal{O}}\}_{i=1}^{N_o}\}$ and a timestep t , and predicts the noise $\boldsymbol{\epsilon}_t$ such that $\mathbf{H}_t = \sqrt{\bar{\alpha}_t} \mathbf{H}_0 + \sqrt{1 - \bar{\alpha}_t} \boldsymbol{\epsilon}_t$. The noisy lane and object latents are first embedded with separate 2-layer MLPs for all i , where $f_{\text{emb},l}$ embeds the lane latents into hidden dimension d_l and $f_{\text{emb},o}$ embeds the object latents into hidden dimension $d_o < d_l$. We apply a sinusoidal positional encoding to each embedded latent vector to mitigate the permutation ambiguity problem

$$\begin{aligned}
\mathbf{l}_i^0 &= f_{\text{emb},l}(\mathbf{h}_{i,t}^{\mathcal{L}}) + \mathbf{p}_i^{\mathcal{L}}, \\
\mathbf{o}_i^0 &= f_{\text{emb},o}(\mathbf{h}_{i,t}^{\mathcal{O}}) + \mathbf{p}_i^{\mathcal{O}},
\end{aligned}$$

where $\mathbf{p}_i^{\mathcal{L}}$ and $\mathbf{p}_i^{\mathcal{O}}$ are the positional encodings of the i -th lane and object, respectively. We then pass the latent embeddings through N_{DM} Factorized Attention Blocks (FABs) with AdaLN-Zero conditioning [239]. Concretely, each FAB in the latent diffusion backbone per-

forms the following operations:

$$\begin{aligned}\mathbb{L}^{k+1} &= \text{O2L}_k(\text{proj}_1^k(\mathbb{O}^k), \mathbb{L}^k, \mathbf{C}), \\ \mathbb{L}^{k+1} &= \text{L2L}_k(\mathbb{L}^{k+1}, \mathbf{C}), \\ \mathbb{O}^{k+1} &= \text{L2O}_k(\text{proj}_2^k(\mathbb{L}^{k+1}), \mathbb{O}^{k+1}, \mathbf{C}), \\ \mathbb{O}^{k+1} &= \text{O2O}_k(\mathbb{O}^{k+1}, \mathbf{C}),\end{aligned}$$

where O2L_k , L2L_k , L2O_k , and O2O_k are DiT blocks [239] with AdaLN-Zero conditioning by a context vector $\mathbf{C}$. The context $\mathbf{C}$ is the sum of (i) a sinusoidal encoding of the diffusion timestep t , (ii) a learnable embedding indicating the scene type (partitioned vs. non-partitioned), and (iii) a learnable “city/domain” embedding—{Singapore, Las Vegas, Boston, Pittsburgh} for nuPlan, and {simulation-compatible, simulation-incompatible} for Waymo. The projections proj_1^k and proj_2^k map object latents to dimension d_ℓ and lane latents to dimension d_o , respectively. After N_{DM} FABs, the lane and object embeddings are decoded with 4-layer MLP heads.

$$\begin{aligned}\hat{\epsilon}_{\mathcal{L},t}^i &= f_\epsilon^l(\mathbf{l}_i^{N_{\text{DM}}}), \\ \hat{\epsilon}_{\mathcal{O},t}^i &= f_\epsilon^o(\mathbf{o}_i^{N_{\text{DM}}}).\end{aligned}$$

Training and Implementation Details

Dataset. The nuPlan dataset contains $\sim 1,300$ hours of logs across four cities (Las Vegas, Boston, Pittsburgh, Singapore). Following [93], we sample 450,000 training frames and 50,000 validation frames using per-city intervals of 30 s (Las Vegas), 1 s (Boston), 2 s (Pittsburgh), and 2 s (Singapore). nuPlan provides vehicles, pedestrians, and static-object classes and three lane types: centerline, green light, and red light. In Scenario Dreamer, traffic-signal phases are represented as lane-aligned vectors that overlap the lane centerlines.

The Waymo Open Motion Dataset comprises 487,002 training scenarios and 44,097 validation scenarios spanning 1,750 km of unique roadway across six U.S. cities (San Francisco, Phoenix, Mountain View, Los Angeles, Detroit, Seattle). For training, we construct initial scenes by sampling a random timestep within each scenario. Following Nocturne [38], scenes are tagged as *simulation-compatible* (no traffic-light signals) or *simulation-incompatible*; this enables classifier-free guidance at inference to favour compatible scenes while still leveraging the full dataset during training. For Waymo, Scenario Dreamer models vehicles, pedestrians, and bicycles, and represents lanes with a single centerline type.

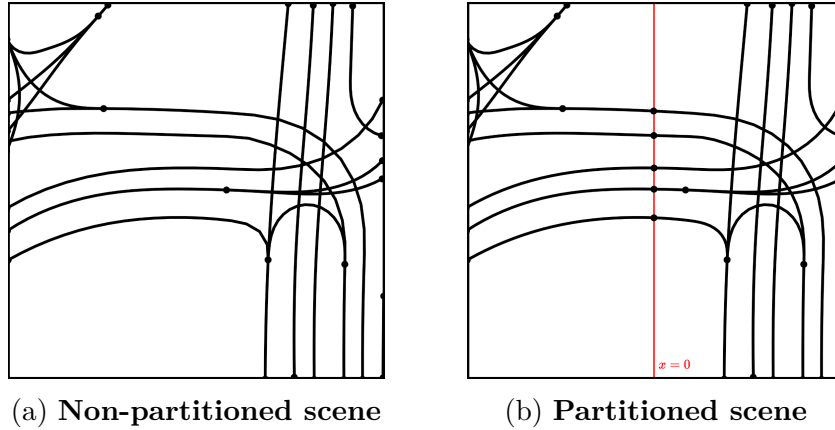


Figure C.7 **Partitioned vs. Non-partitioned Scenes** [44]. In Figure C.7a, we show a non-partitioned scene where lanes can arbitrarily cross the $x = 0$ boundary. In Figure C.7b, we split the lanes that cross the $x = 0$ boundary (shown in red). (© 2025 IEEE)

We preprocess both the nuPlan and Waymo datasets into partitioned and non-partitioned scenes; see Figure C.7 for examples of both formats. For partitioned scenes, we add successor/predecessor connections to lanes split by the $x = 0$ border. Additionally, we follow the lane graph preprocessing approach introduced in SLEDGE [93] where we merge adjacent lanes that are restricted to a single traversable path; this removes all key points in the lane graph with degree = 2. Each traffic scene is processed within a $64\text{m} \times 64\text{m}$ field of view (FOV) centered on the ego agent. We further exclude off-road vehicles, defined as those located more than 1.5m from a lane centerline from the Waymo dataset.

Autoencoder details The autoencoder architecture is shared across datasets with minor dataset-specific adjustments. For nuPlan, we add $\sim 1\text{M}$ parameters to predict lane type (centerline / green-light / red-light), a head omitted for Waymo since global traffic-light states are not modeled there. We train for 50 epochs on a single A100 GPU (≈ 36 hours). Dataset limits are set to $(N_o, N_l) = (30, 100)$ for Waymo and $(61, 100)$ for nuPlan. The model uses $N_E = 2$ encoder and $N_D = 2$ decoder factorized Transformer blocks, hidden sizes of 1024 (lanes), 512 (agents), and 64 (lane connectivity), and latent sizes of 24 (lanes) and 8 (agents). Optimization uses AdamW (learning rate $1\text{e}-4$, weight decay $1\text{e}-4$) with total batch size 128 and a linear decay schedule with 1000 warm-up steps. Loss weights are $\lambda_{\text{lane}} = \lambda_{\text{conn}} = 10$, $\lambda_{\text{num}} = 0.1$, and $\beta = 0.01$. All lane/agent features are min-max normalised to $[-1, 1]$ using training-set statistics. No dropout is applied.

Latent Diffusion Model The latent diffusion model uses $N_{\text{DM}}=2$ factorized Transformer blocks with hidden sizes 2048 (lanes) and 512 (agents). Following [57], we normalise latents using mean/std estimated from a subset of training latents; during training we sample these

normalised latents to form each batch rather than regressing the mean latent as in SLEDGE [93], which we found introduces a beneficial regularization effect. The base model (B) contains one lane-to-lane (L2L) DiT block per FAB, while the large model (L) uses three L2L blocks to allocate more capacity to lane generation; both share the same autoencoder latents. We train with AdamW (lr $1e-4$, weight decay $1e-5$), batch size 1024, and an EMA of 0.9999; the lane noise-prediction loss is scaled by 10. Diffusion uses 100 steps with a cosine variance schedule [280]. At sampling, we apply low-temperature sampling [281] to lane latents with $\alpha=0.75$ and clip noisy lane/object latents $\mathbf{H}_t$ to $(-5, 5)$ after each denoising step. For classifier-free guidance, we drop conditioning with probability 0.1 during training and use guidance scale $s=4.0$ at inference. No dropout is applied.

To address the challenges associated with permutation ambiguity during training, as previously mentioned, we order the lane latents and object latents by minimum x -value. If x -values differ by less than $\epsilon = 0.5$ meters, they are subsequently ordered by minimum y -value, then maximum x -value, and finally maximum y -value, where the maximum is only applied for lanes, since objects are composed of a single point. Figure C.8 illustrates the proposed ordering for the lanes and agents within a given scene. A sinusoidal positional encoding is applied to the embedded lane and object latents to imbue the noisy latents with relative spatial information.

C.3.2 CtRL-Sim Behaviour Model

Architectural Details

The Scenario Dreamer behaviour model builds upon the architectural details of CtRL-Sim, with modifications to support arbitrarily long rollouts without relying on log-replay trajectories. In CtRL-Sim, the goal state for each agent is defined as the final state in the log-replay trajectory and used as conditioning. However, since the Scenario Dreamer initial scene generator provides only the initial agent states, we model agent behaviours directly from these initial states without any notion of goals.

To accommodate multiple agent types (e.g., vehicles, pedestrians, cyclists), we adopt the k -disks tokenization scheme [240], utilizing a discrete vocabulary of size 384, as visualized in Figure C.9. We also remove absolute timestep conditioning, to enable extended rollouts. We adapt its map encoder to process Scenario Dreamer maps in a compact representation. In this representation, adjacent lanes are merged when there is only one traversable path, simplifying the lane structure.

The original CtRL-Sim models the undiscounted return over fixed trajectory lengths T ,

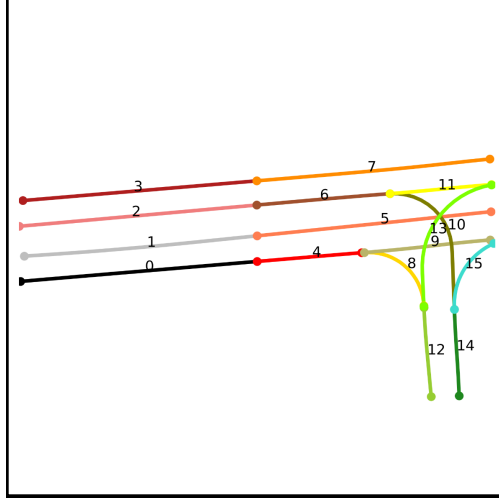


Figure C.8 **Proposed ordering scheme** [44]. We illustrate the proposed ordering scheme for the lanes of a partitioned scene, where agent ordering follows a similar approach. We order first by minimum x -value, and if the difference is less than ϵ , we then order by minimum y -value, then maximum x -value, then maximum y -value. Importantly, all lanes before the $x = 0$ boundary (in $\mathcal{I}_{F_N}$) are assigned an ordering strictly lower than the lanes after the $x = 0$ boundary (in $\mathcal{I}_{F_P}$). (© 2025 IEEE)

defined as $G_t := \sum_{i=t}^T r_i$. However, we found that this approach does not generalize to trajectories longer than T . This significantly limits its applicability to rollouts of arbitrary length. We instead model the *discounted return* over a fixed horizon of $H = 2$ seconds, defined as $G_t := \sum_{i=t}^{t+H} r_i$. This adjustment enhances controllability in practice, as the discounted return focuses on a shorter, more actionable horizon of 2 seconds.

Reward function As mentioned earlier, we aim to produce agent behaviours that challenge the planner. We therefore define the reward function as follows

$$r_t(s_i^t, s_{\text{ego}}^t) = -10 \times \mathbb{1}_{\text{veh-ego coll}}(s_i^t) + \frac{\|s_i^t - s_{\text{ego}}^t\|_2, 10)}{10},$$

where s_{ego}^t is the state of the ego vehicle, or planner, at timestep t , and $\mathbb{1}_{\text{veh-ego coll}}(s_i^t)$ is the indicator function of vehicle i at timestep t colliding with the ego.

Training and Inference Details

Training. We train CtRL-Sim on the full Waymo Open Motion Dataset: 487,002 training scenarios and 44,097 validation scenarios, holding out a subset of *simulation-compatible* validation scenes for testing. It consumes sequences of length $H \times N \times 3$ with horizon $H=32$ and $N=24$ agents; return-to-go components G_t^i are discretised into 350 bins. Follow-

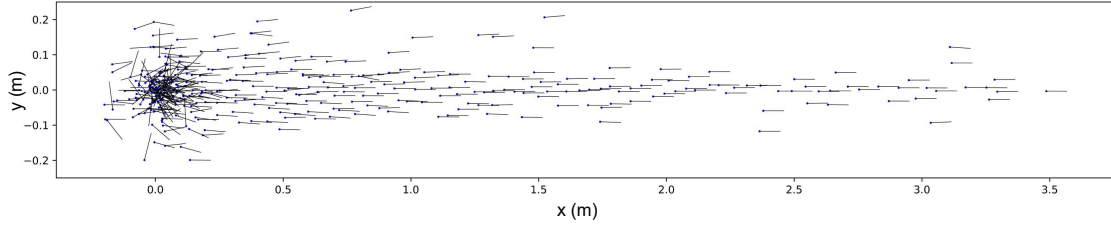


Figure C.9 **Visualization of the k -disks tokenization scheme [44].** Each point denotes the $\Delta x, \Delta y$ offset and $\Delta\theta$ offset denoted by the extending line. (© 2025 IEEE)

ing [109, 111, 240], we encode agents and map in a global frame by centring/rotating to the ego. Agent states match Scenario Dreamer’s object format, with an additional binary ego indicator.

The map context is a set of road segments $\mathcal{M} := \{\mathbf{l}_i\}_{i=1}^{N_l}$, each $\mathbf{l}_i := (p_l^1, \dots, p_l^P)$ with $P=50$ points and connectivity. Because Scenario Dreamer outputs 20-point lanes, we upsample them to 50 for CtRL-Sim. Per sample, we select up to 100 lane segments within a 125×125 m FOV centred on the ego at a random context timestep t , and up to 24 agents within an 80×80 m FOV at t as social context. The model uses hidden size $d=256$, $E=2$ Transformer encoder layers, and $D=4$ decoder layers, and processes all dynamic agents (vehicles, pedestrians, cyclists). Embeddings for missing state/action/return tokens are zeroed; return tokens with horizon truncated to $H' < H$ are also zeroed. Optimisation uses AdamW with total batch size 64 and a learning rate of 5×10^{-4} linearly decayed over 500k steps. The CtRL-Sim architecture has 7.6M parameters and trains in ~ 48 hours on $4 \times \text{A100}$ GPUs.

Inference Recall that although CtRL-Sim is trained with at most $N=24$ agents, at inference it scales to larger scenes by processing 24-agent subsets per timestep. However, the ego must be included in every subset because returns are defined relative to its pose. We use context length $H=32$: at each step the most recent 32 frames are fed, and the scene is normalised to the ego’s position and orientation at the latest context frame. Departing from the original physics-enhanced dynamics, we adopt a delta-based forward model that applies the predicted $(\Delta x, \Delta y, \Delta\theta)$ —derived from the k -disks action vocabulary—directly to the agent state each timestep. This does not enforce strict physical realism but is applicable across multiple agent types and works well in practice [27].

C.4 Evaluation Details

C.4.1 Metrics

Urban Planning Metrics

Urban-Planning metrics. We assess the distributional realism of generated lane-graph connectivity by comparing node-level statistics at *key points* [93, 102, 282]—nodes with degree $\neq 2$ —using 1D Fréchet distances. Key points are shown as black dots in Fig. C.7a. Following [93], we compute the following features on both 50,000 ground-truth test graphs and the generated graphs:

- **Connectivity** — degrees of all key points; list length $50,000 \times \text{avg-num-keypoints-per-graph}$.
- **Density** — number of key points per graph; list length 50,000.
- **Reach** — for each key point, the number of paths to other key points; list length $50,000 \times \text{avg-num-keypoints-per-graph}$.
- **Convenience** — Dijkstra path lengths for all valid key-point pairs; list length $50,000 \times \text{avg-num-paths-per-graph}$.

We report the (unsquared) Fréchet distance (not the squared variant) and, for readability, rescale the Connectivity, Density, Reach, and Convenience distances by $\times 10$, $\times 1$, $\times 1$, and $\times 10$, respectively.

Agent JSD Metrics

Following SceneControl [101], we compute the Jensen-Shannon Divergence (JSD) metrics, which evaluate the distributional realism of the initial vehicle bounding box configurations by comparing 50,000 real and generated scenes. The Jensen-Shannon Divergence between two normalized histograms p and q is computed as

$$\frac{D_{\text{KL}}(p||m) + D_{\text{KL}}(q||m)}{2},$$

where m is the pointwise mean of p and q and D_{KL} is the KL divergence. We compute the following features for computing JSDs:

- **Nearest Distance** — distance from each vehicle to its nearest neighbour; histogram clipped to (0, 50) m with 1 m bins; JSD scaled by 10 for readability.

- **Lateral Deviation** — lateral offset to the closest centerline (vehicles within 1.5 m only); histogram clipped to $(0, 1.5)$ m with 0.1 m bins; JSD scaled by 10.
- **Angular Deviation** — heading difference from the closest centerline (vehicles within 1.5 m only); histogram clipped to $(-200, 200)$ degrees with 5 degrees bins; JSD scaled by 100.
- **Length** — generated vehicle lengths; histogram clipped to $(0, 25)$ m with 0.1 m bins; JSD scaled by 100.
- **Width** — generated vehicle widths; histogram clipped to $(0, 5)$ m with 0.1 m bins; JSD scaled by 100.
- **Speed** — generated vehicle speeds; histogram clipped to $(0, 50)$ m with 1 m bins; JSD scaled by 100.

The total feature list length for computing each above JSD metric is $50,000 \times \text{avg-num-vehicles-per-scene}$.

APPENDIX D DIRECT BEHAVIOUR SPECIFICATION & CONSTRAINED GRPO

D.1 Extension of the approach

D.1.1 Bootstrap Constraint

One challenge that can arise when there are many constraints with the above multiplier normalization scheme is that the coefficient assigned to the main objective in the Lagrangian is defined as

$$\lambda_0 = 1 - \sum_{k=1}^K \lambda_k.$$

As the number of constraints grows, this formulation dilutes the weight and leaves very little (or even no) room for the main objective during the search for a feasible policy. Consequently, the optimization process may have limited capacity to improve on the main task until feasibility is achieved. Moreover, when additional constraints are introduced, the optimization path can become discontinuous between different feasible regions, further hindering progress on the main objective.

A natural solution is to endow the main objective with the same status as the behavioural constraints that are being enforced. This can be achieved by defining an additional function $S_{K+1}(s, a)$ that captures some notion of success on the primary task. Many reinforcement learning tasks are, in fact, defined in terms of such sparse and well-specified success conditions, which are typically complemented with a dense reward function to guide exploration toward these conditions [283].

We propose the usage of a *success constraint* of the form

$$J_{S_{K+1}}(\pi) \geq \tilde{d}_{K+1},$$

where S_{K+1} is implemented as an indicator cost function, analogous to those described earlier, and add it to the existing set of behavioural constraints

$$\{J_{C_k}(\pi) \leq \tilde{d}_k\}_{k=1}^K.$$

While using the success constraint alone can support learning progress on the main task, it typically provides only a sparse signal, which may be difficult to discover in challenging domains.

Since the success function S_{K+1} is intended to be highly correlated with the reward function R , we can go a step further and *replace* the reward multiplier λ_0 with the success multiplier λ_{K+1} when enforcing this constraint. This allows the optimization process to leverage the dense reward signal while still emphasizing success as a behavioural requirement. However, to preserve a genuine maximization objective over the original reward, we retain λ_0 when the other constraints are already satisfied, so that maximum progress can be made on $J_R(\pi)$.

Concretely, we take the larger of the two coefficients for weighting the main objective:

$$\tilde{\lambda}_0 := \max(\lambda_0, \lambda_{K+1}),$$

and substitute λ_0 with $\tilde{\lambda}_0$ in Equation 7.13. In this framework, the $(K + 1)$ -th constraint is referred to as a *bootstrap constraint*.

This approach of encoding the success criterion as part of the constraint set can be interpreted as a way to relax the behavioural constraints during optimization without modifying the underlying convergence requirements. For comparison, Calian et al. [154] propose adjusting the learning rates of the Lagrange multipliers to automatically soften constraints that remain unsatisfied after a certain period. In contrast, the bootstrap constraint enables progress on the main task from the outset, without converting hard constraints into soft ones.

D.1.2 Constrained SAC

In this section, we illustrate how our algorithm can be integrated with the Soft Actor-Critic (SAC) framework in Algorithm 6. The exact values of all hyperparameters used in our experiments are reported in Tables D.1 and D.2.

An important difference between the standard, unconstrained Soft Actor-Critic algorithm [59] and our constrained version lies in the update schedule. In the unconstrained setting, SAC is typically updated after every environment step to maximize sample efficiency. In the constrained setting, however, the constraint objectives must be optimized on-policy. Updating the SAC agent after every environment step would therefore yield only one-sample estimates of the Lagrange multipliers' objectives, leading to very high variance. Conversely, freezing the SAC agent for as many environment steps as the multiplier batch size N_λ would substantially degrade the overall sample efficiency of the algorithm.

One option would be to relax the on-policy requirement of the multiplier updates. However, in preliminary experiments we observed that updating the multipliers very frequently while using a large replay buffer—containing many transitions generated by previous policies—leads to significant overshooting and instability, ultimately harming convergence. This highlights

Algorithm 6 SAC-Lagrangian with Bootstrap Constraint

Require: learning rate β , replay buffer $\mathcal{B}$, entropy coefficient α , minibatch sizes N_θ and N_λ

Require: Initialize policy π_θ and value functions $Q_\phi^{(k)}$ randomly, $k = 0, \dots, K + 1$

Require: Initialize Lagrange multiplier parameters z_k

Require: Collect transitions to fill $\mathcal{B}$ with $\max(N_\theta, N_\lambda)$ samples

```

1: for updates  $u = 1, 2, \dots$  (until convergence) do
2:   # Data collection
3:   Sample from current policy:  $a \sim \pi_\theta(\cdot | s)$ 
4:   Interact with environment to get next state, reward and indicators:  $(s', r, \{e^{(k)}\}_{k=1}^{K+1})$ 
5:   Append transition  $(s, a, r, s', \{e^{(k)}\}_{k=1}^{K+1})$  to  $\mathcal{B}$ 
6:   # Policy gradient update
7:   if  $u \bmod M_\theta = 0$  then
8:     Sample a minibatch of  $N_\theta$  transitions uniformly from  $\mathcal{B}$ 
9:     Sample next actions  $a'_i \sim \pi_\theta(\cdot | s'_i)$  ( $i = 1, \dots, N_\theta$ )
10:    for  $k = 0, \dots, K + 1$  do
11:      Set “rewards”:  $r_i^{(0)} \leftarrow r_i, \quad r_i^{(k)} \leftarrow e_i^{(k)}$ 
12:      Compute targets:  $y_i^{(k)} \leftarrow -\alpha \log \pi_\theta(a'_i | s'_i) + \min_{j \in \{1, 2\}} Q_{\phi_j}^{(k)}(s'_i, a'_i)$ 
13:      Update critics (Adam descent):

$$\nabla_{\phi_j} \frac{1}{N_\theta} \sum_{i=1}^{N_\theta} \|Q_{\phi_j}^{(k)}(s_i, a_i) - (r_i^{(k)} + (1 - \text{done}_i) \gamma y_i^{(k)})\|_2$$

14:    end for
15:    Re-sample current actions  $a_i \sim \pi_\theta(\cdot | s_i)$ 
16:    Update policy (Adam ascent):

$$\nabla_\theta \frac{1}{N_\theta} \sum_{i=1}^{N_\theta} \left[ -\alpha \log \pi_\theta(a_i | s_i) + \max(\lambda_0, \lambda_{K+1}) \min_j Q_{\phi_j}^{(0)}(s_i, a_i) \right. \\ \left. + \lambda_{K+1} \min_j Q_{\phi_j}^{(K+1)}(s_i, a_i) - \sum_{k=1}^K \lambda_k \min_j Q_{\phi_j}^{(k)}(s_i, a_i) \right]$$

17:  end if
18:  # Multipliers update
19:  if  $u \bmod M_\lambda = 0$  then
20:    Take a minibatch of the most recent  $N_\lambda$  transitions from  $\mathcal{B}$ 
21:    for  $k = 0, \dots, K + 1$  do
22:      Compute average costs:  $\tilde{J}_{C_k}(\pi) \leftarrow \frac{1}{N_\lambda} \sum_{i=1}^{N_\lambda} e_i^{(k)}$ 
23:      Update  $z_k$  (Adam descent):

$$\begin{cases} \nabla_{z_k} [\lambda_k (\tilde{J}_{C_k}(\pi) - \tilde{d}_k)], & \text{if } k = K + 1, \\ \nabla_{z_k} [\lambda_k (\tilde{d}_k - \tilde{J}_{C_k}(\pi))], & \text{otherwise.} \end{cases}$$

24:    end for
25:  end if
26: end for

```

a trade-off between three factors: (i) the variance of the multiplier objective estimates, (ii) the degree to which these updates remain on-policy, and (iii) the overall sample efficiency of the algorithm. In practice, we found that the update frequencies M_θ and M_λ reported in Tables D.1 and D.2 provide a good balance between these factors.

Another important implementation detail is that we maintain $K + 1$ separate critic networks to model the discounted expected returns of both the reward and each cost component. Specifically, $Q^{(0)}$ is used for the main reward objective, while $Q^{(k)}$ for $k = 1, \dots, K + 1$ correspond to the critics associated with each constraint in the Lagrangian. Using separate critics prevents rapid changes in the effective scale of the objective, as seen by the critics, when the multipliers λ_k are adjusted. Each critic can therefore focus on modeling the agent’s behavior with respect to its specific signal (reward or cost), leading to more stable learning.

D.2 Experiments with Constrained SAC: Arena environment

In this section, we train constrained SAC agents [59] to solve navigation tasks with up to 5 constraints imposed on their behaviour. Many of these constraints interact both with the main task and with one another, which substantially restricts the space of admissible policies. To evaluate the effectiveness of the proposed framework in specifying complex behaviours under multiple simultaneous constraints, we conduct the majority of our experiments in the *Arena* environment (Figure 7.1, left). This environment allows us to systematically assess how easily desired behaviours can be specified and how well the algorithm handles a large number of concurrent constraints.

D.2.1 Environment details

In the *Arena* environment, the agent must reach a green tile (Fig. 7.1, left). We train with constraints $\{On-Ground, Not-in-Lava, Looking-At-Marker, Under-Speed-Limit, Above-Energy-Limit\}$. Observations include XYZ position, heading and velocity; goal pose relative to the planner frame and its Euclidean distance; and an on-ground flag. For *Looking-At-Marker* the agent also receives its XZ look vector, Y-angular velocity, the marker’s relative XZ position and distance, the normalized looking–marker angle, and a *marker-within-field-of-view* indicator (fixed cone). For *Energy* it receives normalized energy value and an indicator saying if it is currently recharging. For *Lava* it receives an in-lava indicator plus 25 vertical-raycast indicators indicating in-lava=1, in-safe-tile=0. We append per-episode constraint-violation rates and normalized remaining time; the agent’s complete observation vector has a dimensionality of 53. The action space has five continuous scalars in $[-1, 1]$: v_x , v_z , yaw rate,

jump, and recharge. A reward of 1 is given on reaching the goal (terminal) and 0 otherwise, with a small distance-based shaping term [283].

We also add to the agent’s observations the per-episode rates of indicator cost functions to the agent observation for each of the constraint as well a normalised representation of the remaining time-steps before reaching the time limit condition. This results in a total dimensionality of 53 for the observation vector. The action space is composed of 5 continuous actions (clamped between -1 and 1) which represent its XZ velocity and Y-angular velocity, a jump action (jump is triggered when the agent outputs a value above 0 for that dimensionality) and a recharge action (also with threshold of 0). The reward function is simply 1 when the agent reaches the goal (causing termination), 0 otherwise, and augmented with a small shaping reward function [283] based on whether the agent got closer or further away from the goal location.

D.2.2 Hyper-parameters

Most of the hyper-parameters are the same as in the original unconstrained Soft Actor-Critic (SAC) [59]. Some additional hyper-parameters emerge from the constraint enforcement aspect of our version of SAC-Lagrangian and are described in the Algorithm section above. An Adam optimizer [284] is used for all parameter updates (policy, critics and Lagrange multipliers). For all experiments taking place in the Arena Environment, the policy is parameterized as a two layer neural networks that outputs the parameters of a Gaussian distribution with a diagonal covariance matrix. The hidden layers are composed of 256 units and followed by a *tanh* activation function. The first hidden layer also uses layer-normalisation before the application of the *tanh* function. As we already noted, we use $K + 1$ fully independent critic models to estimate the expected discount sum of each of the constraints and of the main reward function. The critic models are also parameterized with two-hidden-layers neural networks with the same size for the hidden layers as the policy but instead followed by *relu* activation functions. Table D.1 shows the hyper-parameters used in our experiments conducted in the Arena environment.

D.2.3 Experiments

Multiplier Normalization Our first set of experiments examines the impact of *normalizing* the Lagrange multipliers. To highlight this effect, we designed a simple illustrative scenario in which one of the constraints remains unsatisfied for an extended period of time. Concretely, the agent is tasked with meeting the impossible constraint of never touching the ground for the first 7.5M environment steps, after which the constraint is relaxed. As shown

in Figure D.1 (red), the multiplier associated with the unsatisfied constraint grows without bound. While this initially leads to faster convergence on the main task, it eventually destabilizes the entire learning process with the critic loss diverging and overall performance collapsing. In contrast, when applying our normalization technique (Figure D.1, blue), both the multiplier and critic losses remain bounded, effectively preventing these instabilities.

Single Constraint satisfaction We leverage our framework to encode various behavioural preferences as indicator functions, each associated with a specified threshold. Figure D.2 shows that our SAC-Lagrangian algorithm with multiplier normalization is able to solve the navigation task while satisfying the imposed behavioural requirements when each constraint is applied individually. Notably, the impact of different constraints on main task is not uniform: some constraints, such as avoiding jumps, have little effect and allow the agent to solve the task quickly, whereas others, such as avoiding specific terrain types (e.g., lava), substantially increase the difficulty of the navigation problem.

Table D.1 **Hyper-Parameters for Arena Experiments [25]**. We present the hyper-parameters used for experiments in the Arena Environment.

General	DISCOUNT FACTOR γ	0.9
	NUMBER OF RANDOM EXPLORATION STEPS	10000
	NUMBER OF BUFFER WARMUP STEPS	2560
SAC Agent	LEARNING RATE β	0.0003
	TRANSITIONS BETWEEN UPDATES M_θ	200
	BATCH SIZE N_θ	256
	REPLAY BUFFER SIZE	1,000,000
	INITIAL ENTROPY COEFFICIENT α	0.02
	TARGET NETWORKS SOFT-UPDATE COEFFICIENT τ	0.005
Lagrange Multipliers	LEARNING RATE β	0.03
	INITIAL MULTIPLIER PARAMETERS VALUE z_k	0.02
	TRANSITIONS BETWEEN UPDATES M_λ	2000
	BATCH SIZE N_λ	2000
Constraint Thresholds	HAS REACHED GOAL (LOWER-BOUND)	0.99
	NOT LOOKING AT MARKER	0.10
	NOT ON GROUND	0.40
	IN LAVA	0.01
	ABOVE SPEED LIMIT	0.01
	IS UNDER THE MINIMUM ENERGY LEVEL	0.01

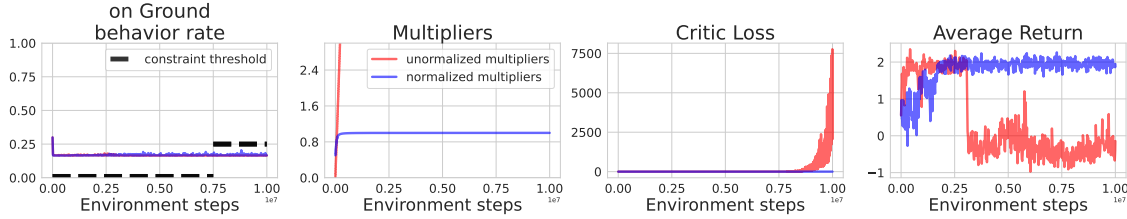


Figure D.1 **Effects of Multiplier Normalization in the Arena Environment [25]**. The multiplier normalization keeps the learning dynamics stable when discovering a constraint-satisfying behaviour takes a large amount of time. To simulate such a case, an impossible constraint is set for 7.5M steps and then replaced by a feasible one for the last 2.5M steps. The method using unnormalized multipliers (red) keeps taking larger and larger steps in policy space leading to the divergence of its learning dynamics and complete collapse of its performance.

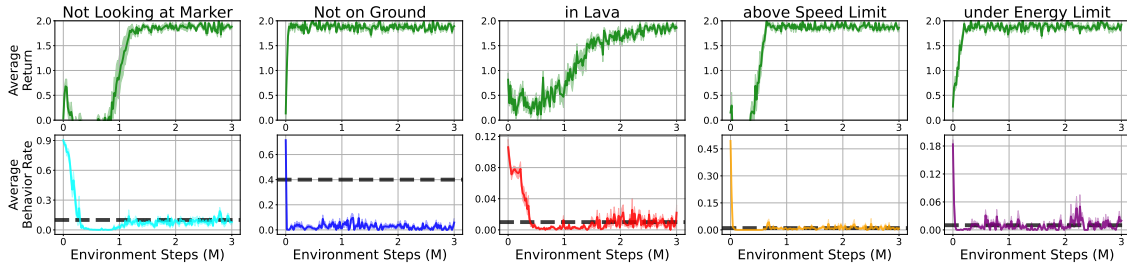


Figure D.2 **Single Behaviour Specification in the Arena Environment [25]**. Each *column* presents the results for an experiment in which the agent is trained for 3M steps with a *single constraint* enforced on its behaviour. Training is halted after every 20,000 environment steps and the agent is evaluated for 10 episodes. All curves show the average over 5 seeds and envelopes show the standard error around that mean. The top row shows the average return, the bottom row shows the average behaviour rate on which the constraint is enforced. The black dotted lines mark the constraint thresholds.

Multiple Constraints Satisfaction In Figure D.3, we first show that an unconstrained SAC agent can indeed learn to navigate properly while not respecting any of the constraints. This assures us that none of the constraints are trivial to solve. The next row shows that when all constraints are imposed simultaneously, the agent is correctly able to learn a feasible policy but cannot simultaneously perform well on the main navigation task. In this setting, the agent converges to a trivial behaviour that focuses exclusively on satisfying the constraints, from which it is difficult to escape without violating them. By introducing a *success constraint* (row (c)), the agent is able to satisfy all behavioural constraints while also achieving success on the navigation task at convergence. The addition of the success constraint promotes exploration of regions of the policy space that would otherwise remain infeasible, enabling the discovery of feasible solutions with significantly better task performance. The best results are obtained

when using the success constraint as a *bootstrap constraint*, effectively transferring the weight λ_{K+1} to the main reward during the search for a feasible policy.

D.3 Experiments with Constrained SAC: OpenWorld Environment

In this section, we further test the scalability of our approach in the *OpenWorld* environment (Figure 7.1, right), a significantly larger and more diverse map generated using the GameR-Land map generator [285]. This experiment is designed to examine whether our framework remains effective in richer and more realistic settings that are representative of industrial applications for agent behaviour specification in video games.

D.3.1 Environment details

The OpenWorld environment is a large map (about $30,000\times$ the agent’s size) with multi-storey buildings and staircases, mountains, tunnels, natural bridges, and lava. It contains 50 jump pads that launch the agent on contact. At the start of each episode, a goal is placed randomly; the agent must navigate to it while also keeping gaze on a designated marker. The action vector has five scalars: planar translation in the XY frame (2), jump (1), view-rotation controlling the look direction independent of travel (1), and recharge (1); notably, recharging immobilises the agent.

The agent receives as observation at every timestep: (i) its XYZ position relative to the goal, and its normalised velocity and acceleration; (ii) relative position to the nearest jump pad; (iii) marker features—relative XZ position and distance, the normalised look-marker angle, and a *marker-within-field-of-view* indicator (fixed cone); (iv) energy features—current energy, an indicator for “recharging now,” and a boolean flag for “recharged on the previous step”; (v) several additional constraint indicators — standing in lava, on ground, and below the minimum energy level; and (vi) two 8×8 raycast channels around the agent to sense (1) lava, and (2) other collidable geometry (e.g., buildings, doors, mountains).

D.3.2 Hyper-parameters

The SAC agent in the OpenWorld environment uses the same architecture and similar hyper-parameters as in [168]. The raycasts and raw state described above are processed using two separate embedding models. For the raycasts, we employ a CNN with 3 convolutional layers, each with a corresponding ReLU layer. The raw state is processed using a separate 3-layer MLP with 1024 hidden units at each layer. The two representations are concatenated into a single vector representing the current state. The policy is parameterized by a 3-layer

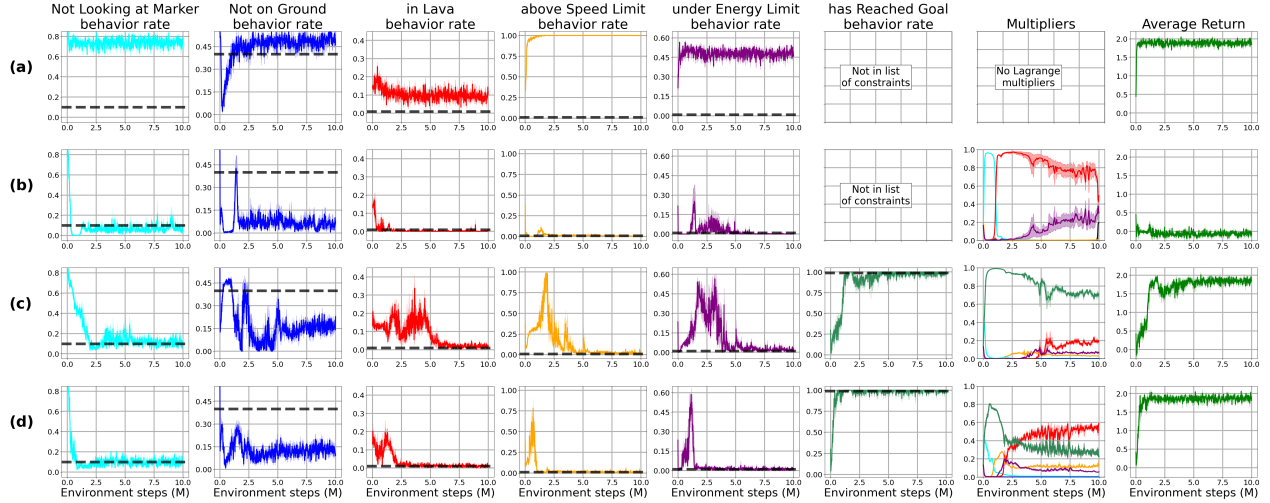


Figure D.3 **Multiple Behaviour Specification in the Arena Environment [25]**. Each row presents the results of an experiment in which an agent is trained for 10M steps. Training is halted after every 20,000 environment steps and the agent is evaluated for 10 episodes. All curves show the average over 5 seeds and envelopes show the standard error around that mean. **(a)** Unconstrained SAC agent; none of the behavioural preferences are enforced and consequently improvement on performance is very fast but none of the constraints are satisfied. **(b)** SAC-Lagrangian with the 5 behavioural constraints enforced. While each constraint was successfully dealt with when imposed one by one (see Figure D.2), maximising the main objective when subject to all the constraints *simultaneously* proves to be much harder. The agent does not find a policy that improves on the main task while keeping the constraints in check. **(c)** By using an additional success constraint (that the agent should reach its goal in 99% of episodes), the agent can cut through infeasible policy space to start improving on the main task and optimise the remaining constraints later on. **(d)** By using the success constraint as a bootstrap constraint (bound to the main reward function) improvement on the main task is much faster as the agent benefits from the dense reward function to improve on the goal-reaching task.

MLP that receives as input the concatenated representation and outputs the parameters of a Gaussian distribution with a diagonal covariance matrix. Each hidden layer is composed of 1024 hidden units and is followed by a ReLU activation function. The critic models are also parameterized by 3-layer MLP, are composed of 1024 hidden units and use ReLU activation functions. Table D.2 shows some of these hyper-parameters with a focus on the constrained enforcement aspect of our version of SAC-Lagrangian.

Table D.2 **Hyper-Parameters for OpenWorld Experiments [25]**. We present the hyper-parameters used for experiments in the OpenWorld Environment.

General	DISCOUNT FACTOR γ	0.99
	NUMBER OF RANDOM EXPLORATION STEPS β	200
	NUMBER OF BUFFER WARMUP STEPS β	2560
SAC Agent	LEARNING RATE β	0.0001
	BATCH SIZE N_θ	2560
	REPLAY BUFFER SIZE	4,000,000
	INITIAL ENTROPY COEFFICIENT α	0.005
	TARGET NETWORKS SOFT-UPDATE COEFFICIENT τ	0.005
Lagrange Multipliers	LEARNING RATE β	0.00005
	INITIAL MULTIPLIER PARAMETERS VALUE z_k	0.02
	TRANSITIONS BETWEEN UPDATES	EVERY TIMESTEP
	BATCH SIZE N_λ	5000
Constraint Thresholds	HAS REACHED GOAL (LOWER-BOUND)	0.80
	NOT LOOKING AT MARKER	0.10
	NOT ON GROUND	0.40
	IN LAVA	0.001
	IS UNDER THE MINIMUM ENERGY LEVEL	0.01

D.3.3 Experiments

In the OpenWorld environment, we evaluate whether the proposed framework scales effectively to more complex and realistic tasks. Unlike the *Arena* environment, *OpenWorld* features uneven terrain, buildings, and interactable elements such as jump pads, making this setting much closer to real-world reinforcement learning applications in the video game industry. For this experiment, we trained a SAC-Lagrangian agent to solve the navigation problem while adhering to four behavioural constraints: *On-Ground*, *Not-In-Lava*, *Looking-At-Marker*, and *Above-Energy-Limit*. The SAC component uses the same hyperparameters as in [168]. The results, shown in Figure D.4, indicate that although training in this larger and more complex environment requires up to 50 million environment steps, the agent suc-

cessfully completes the task while respecting all constraints. These findings provide strong evidence that the proposed framework scales favourably to more challenging environments for direct behaviour specification.

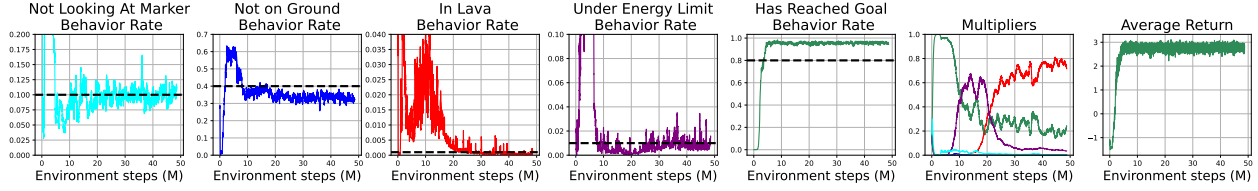


Figure D.4 **Multiple Behaviour Specification in the OpenWorld Environment [25].** A SAC-Lagrangian agent trained to solve the navigation problem in the OpenWorld environment while respecting four constraints and imposing the bootstrap constraint. Results suggest that our SAC-Lagrangian method using indicator cost functions, normalised multipliers and bootstrap constraint scales well to larger and more complex environments.

D.4 Additional Details on Constrained GRPO

D.4.1 Numerical Example of Theorem 4.4.1

We provide a simple 1-constraint example to understand the difference in practice. Let $\lambda_R = 0.3$ and $\lambda_C = 0.7$ (since both must sum to 1), $\sigma_R = 0.5$, $\sigma_C = 0.2$, and $\rho = 0.1$. Then we can compute the standard deviation of the combined signal with scalarized rewards as

$$\begin{aligned}\sigma_{R_S} &= \sqrt{(0.3^2 \cdot 0.5^2) + (0.7^2 \cdot 0.2^2) + (2 \cdot 0.3 \cdot 0.7 \cdot 0.1 \cdot 0.5 \cdot 0.2)} \\ &\approx 0.215174.\end{aligned}$$

With σ_{R_S} , the effective weights on (Z_R, Z_C) :

Scalarized Advantages: $(0.3, 0.7)$,

Scalarized Rewards: $\left(\frac{0.3 \cdot 0.5}{0.215174}, \frac{0.7 \cdot 0.2}{0.215174}\right) \approx (0.697, 0.651)$.

As we can see from this simple example, the reward term was 2.33 times less important according to the current Lagrangian multiplier. However, the 2nd approach effectively scales its coefficient to be even more important than the constraint.

With this, we conclude that if you want λ to faithfully encode your intended reward–constraint trade-off independently of group scales and covariance, the per-component standardization followed by combining terms using λ is the more faithful approach. In general,

the second approach is not invalid but has the caveat that it implicitly reweighs the multipliers by the components’ standard deviations (and ρ). This results in agents effectively having to satisfy the high-variance components first prior to satisfying the low-variance ones, *regardless of the Lagrangian multiplier λ_i values*. In our experiments in Section 7.5.2, we present results on a toy environment and in the real-world NAVSIM driving benchmark, comparing both approaches with fixed and varying Lagrangian multipliers.

D.4.2 Environment Details

Our gridworld is a 10×10 grid in which the agent is initialized uniformly at random and must navigate to a randomly sampled goal location. Reaching the goal yields a sparse terminal reward of 1.0. At the start of each episode, 20% of the tiles are designated as lava; stepping onto a lava tile incurs a penalty of 1.0 (i.e., a cost). The agent is also equipped with a battery that drains by 5% at every timestep and incurs a penalty whenever its charge falls below 10%. The action space consists of five discrete actions: moving in the four cardinal directions, or staying in place to recharge at a rate of 20% per timestep. The agent observes a compact state representation consisting of:

- its (x, y) position, normalized to $[0, 1]^2$;
- the goal (x, y) coordinates, normalized to $[0, 1]^2$;
- the current battery level in $[0, 1]$;
- a 5×5 local occupancy map centered on the agent, where entries indicate lava tiles (flattened to a 25-dimensional vector).

Episodes last at most 80 timesteps and terminate early only upon reaching the goal.

D.4.3 Implementation Details

We train a categorical policy parameterized by a two-layer MLP (hidden size 128 with ReLU activations) that outputs action logits. Policies are optimized with the GRPO clipped objective ($\epsilon = 0.2$) using 2 epochs per update, minibatches of size 2048, and an entropy bonus of 0.001. Training runs for 8000 update steps; each update collects 64 episodes via grouped rollouts (group size 8 across 8 groups). We use Adam for the policy optimizer (learning rate 5×10^{-4} by default) and, when constraints are enabled, a separate Adam optimizer for cost weights (learning rate 10^{-2}). Cost weights are parameterized via a softmax over {reward, lava, battery} parameters and updated every step and initialized with logits of 0.02.

D.4.4 Additional Experiments

We include additional results comparing scalarizing rewards with scalarizing advantages for a single constraint in Figure D.5, and with both battery and lava constraints in Figure D.6. We refer the reader to the caption for more details.

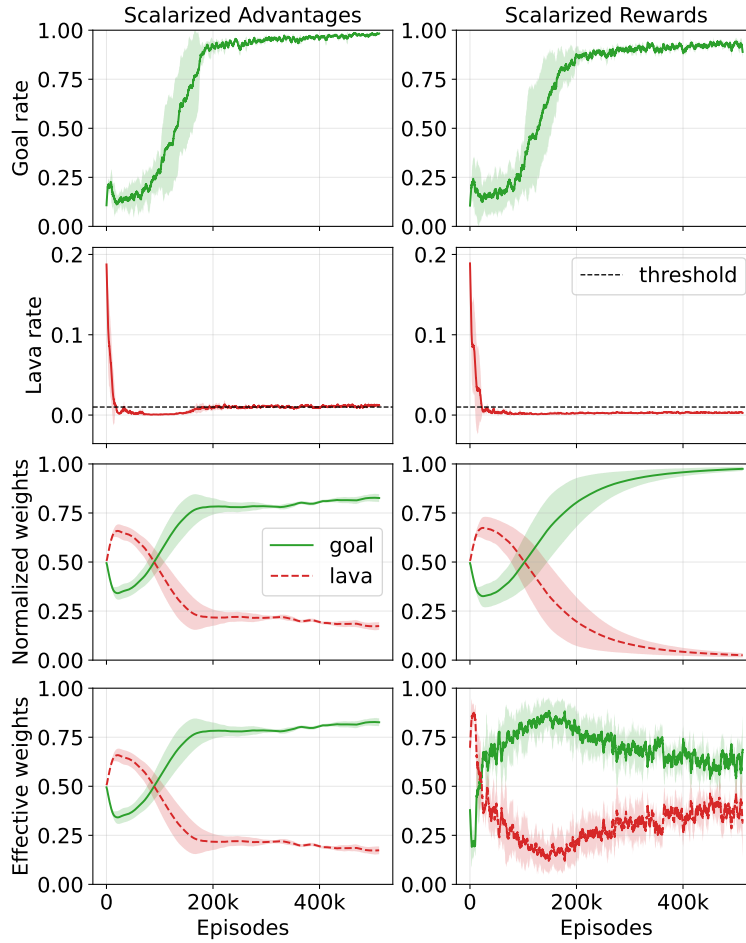


Figure D.5 **Learning Dynamics of Constrained GRPO** with $\tilde{d}_{\text{lava}} = 0.01$ on the grid-world, comparing *scalarized advantages* (left) and *scalarized rewards* (right). In this experiment, we set $\tilde{d}_{\text{lava}} = 0.01$ and do not enforce the battery constraint. Scalarized advantages again produces a stronger final policy and makes effective use of the allowable cost budget, with the lava rate touching the specified threshold in order to allow for success on the main task. In contrast, scalarized rewards drives the constraint rate to near-zero early in training; despite small multiplier values, the corresponding effective weights remain larger and noisier, leading to less stable learning and lower final performance.

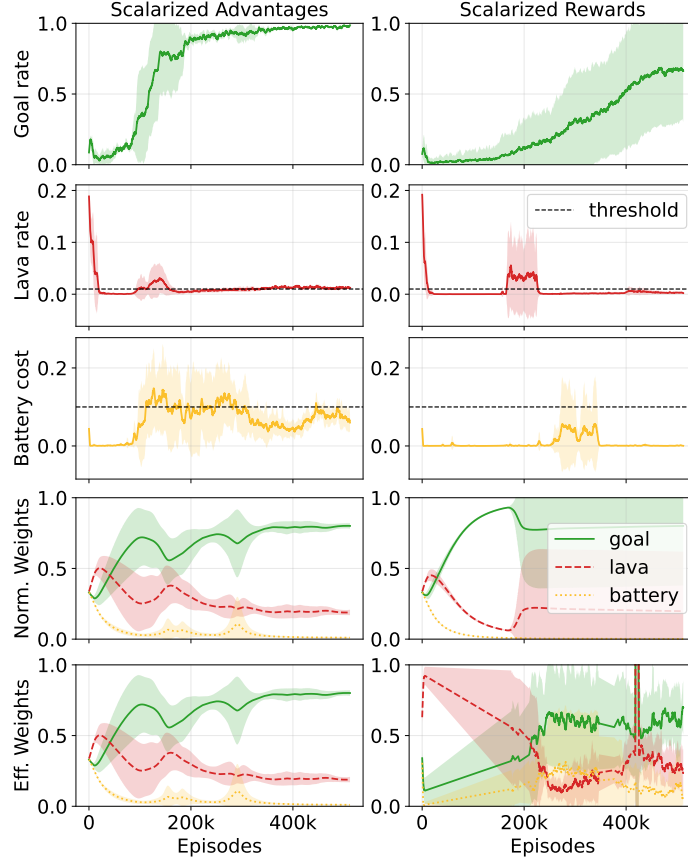


Figure D.6 **Learning Dynamics of Constrained GRPO with $\tilde{d}_{\text{lava}} = 0.01$ and $\tilde{d}_{\text{battery}} = 0.1$** on the gridworld, comparing *scalarized advantages* (left) and *scalarized rewards* (right). In this experiment, we set $\tilde{d}_{\text{lava}} = 0.01$ and $\tilde{d}_{\text{battery}} = 0.1$. Scalarized advantages again produces a stronger final policy and makes effective use of its allowable costs' budget, with both the lava rate and battery rate stabilizing near their specified thresholds in order to allow for success on the main task. In contrast, scalarizing rewards drives both constraints' cost rates to near-zero early in training. Even with small multiplier values, the corresponding effective weights remain larger and noisier, leading to less stable learning and lower final performance. Interestingly, we see that when it tries to creep higher, the effective multipliers react excessively, resulting in the agent not adequately navigating the environment, as shown by the poor goal success rate.