A Language-Guided Generative Data Engine for Autonomous Driving

Brian Yang

CMU-RI-TR-26-16

March 30, 2026

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Katerina Fragkiadaki, *co-chair*
Jeff Schneider, *co-chair*
Maxim Likhachev
Philipp Krähenbühl (UT Austin)

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in Robotics.

Copyright © 2026 Brian Yang. All rights reserved.

Abstract

Autonomous driving systems struggle with the long tail of rare and safety-critical scenarios that are sparsely represented in real-world data. While recent learning-based planners perform well on common situations, their reliability degrades in edge cases, and existing approaches based on simulation or world modeling remain costly and difficult to scale.

This thesis proposes a complementary paradigm for scaling driving policies by leveraging language supervision and generative models as a unified data engine. Rather than relying solely on logged demonstrations, we use vision-language models to infer structured constraints and reward functions from high-level specifications of desired behavior. These programmatic rewards are then used to guide trajectory diffusion models, enabling the generation of diverse, physically consistent, and behaviorally targeted driving trajectories.

We first show that combining pretrained diffusion policies with reward guidance enables zero-shot instruction following, without requiring language-annotated driving data. We then demonstrate how guided diffusion can be used as a scalable offline data generator, producing counterfactual trajectories that expand coverage of rare and safety-critical scenarios. Policies distilled from this synthetic data outperform those trained purely on logged datasets and no longer require test-time guidance. Finally, we close the loop by learning reward programs automatically via vision-language feedback, enabling iterative improvement of both the data generator and the policy.

Together, this work presents a scalable framework in which language specifies what should happen, and generative models produce how it happens, yielding a practical path toward robust driving policies without exhaustive real-world data collection or high-fidelity simulation.

Contents

1 Introduction **1**

2 Diffusion-ES: Gradient-free Planning with Diffusion for Autonomous Driving and Zero-Shot Instruction Following **5**
2.1 Introduction . . . 5
2.2 Related work . . . 8
2.3 Method . . . 10
2.3.1 Background . . . 10
2.3.2 Diffusion-ES . . . 11
2.3.3 Mapping language instructions to reward functions with LLM prompting . . . 13
2.4 Experiments . . . 14
2.4.1 Closed-loop driving . . . 15
2.4.2 Language instruction following . . . 17
2.4.3 Lane following . . . 19
2.4.4 Runtime analysis . . . 20
2.5 Conclusion . . . 20

3 GANDALF: Language-Controllable Driving without Labels via Diverse Reward-Guided Synthesis **23**
3.1 Introduction . . . 23
3.2 Related work . . . 26
3.3 Scene-Conditioned Flow Prior . . . 27
3.3.1 Overview . . . 27
3.3.2 Model Architecture . . . 28
3.3.3 Reward-Guided Inference . . . 29
3.4 Counterfactual Synthesis and Distillation . . . 29
3.4.1 General Driving . . . 30
3.4.2 Controllability . . . 31
3.5 Experiments . . . 32
3.5.1 Implementation Details . . . 32
3.5.2 General Driving Performance . . . 33
3.5.3 Controllability Tasks . . . 34
3.5.4 Visually-Grounded Controllability Tasks . . . 35

3.5.5 Diversity–Quality Tradeoff . . . 37
3.6 Limitations and Conclusion . . . 38

4 EurekaDrive: Vision-Language Guided Program Search for Language-Conditioned Driving 41
4.1 Introduction . . . 41
4.2 Related Work . . . 43
4.2.1 End-to-end autonomous driving . . . 43
4.2.2 Language model programs for robotics . . . 44
4.2.3 Language datasets for autonomous driving . . . 44
4.3 Background . . . 45
4.3.1 Problem setting . . . 45
4.3.2 Diffusion-ES . . . 45
4.4 Method . . . 46
4.4.1 VLM-guided program search . . . 46
4.4.2 Data collection . . . 48
4.4.3 Policy finetuning . . . 49
4.5 Experiments . . . 50
4.5.1 Benchmark and Experimental Setup . . . 50
4.5.2 Program Search for Maneuver Generation . . . 51
4.5.3 Training Language-Conditioned Driving Agents . . . 53
4.5.4 Program Search for General Driving Improvement . . . 56
4.6 Conclusion . . . 58

5 Latent Preference Finetuning for Driving Policies 59
5.1 Introduction . . . 59
5.2 Related Work . . . 60
5.3 Method . . . 61
5.3.1 Model Architecture . . . 61
5.3.2 Intent Policy Finetuning . . . 62
5.4 Results . . . 62

6 Conclusion 63
6.1 Summary of Contributions . . . 63
6.2 Limitations . . . 65
6.3 Future Directions . . . 65
6.4 Broader Impact and Closing Remarks . . . 67

Bibliography 69

List of Figures

2.1 Overview for Diffusion-ES . *Left:* Generative evolutionary search with Diffusion-ES . Trajectories are colored by rewards (blue is low, green is high). *Right:* We visualize the mutations for varying noise levels. Color denotes timestep along trajectory. While noise perturbations alone can lead to unrealistic trajectories, denoising helps project samples back onto the trajectory data manifold. 10

2.2 Shaping reward functions with language instructions using LLM prompting and optimizing them with Diffusion-ES . For the same initial scene, we show two distinct language instructions, along with their generated reward shaping code and generated trajectories from Diffusion-ES . . 13

2.3 Following driving instructions. Diffusion-ES outperforms all baselines in optimizing complex language-shaped reward functions. 18

3.1 GANDALF generates diverse driving demonstrations by guiding a scene-conditioned flow prior with reward functions over logged scenes. By varying the reward specification—general driving quality, task-specific objectives (e.g., "change lanes right"), or visually-grounded instructions (e.g., "overtake the gray van")—we produce counterfactual trajectories that are task-diverse, reward-labeled, and language-labeled, without new data collection or human annotation. 25

3.2 Qualitative examples of reward-guided controllability. Each row shows the same scene under different reward functions. **Top:** lane change rewards steer the ego vehicle left or right (red) relative to the baseline trajectory (green). **Bottom:** the accelerate reward increases forward progress, and the follow vehicle reward closes the gap to the lead vehicle (highlighted in red). 36

3.3 PDM score vs. sampling temperature across 300 NAVSIM scenes. At low temperatures, both approaches converge as samples become near-identical. As temperature increases, the flow prior degrades sharply while GANDALF remains stable. Error bars show standard error of the mean. 38

4.1 Given a driving scenario, high-level language goal, and associated reward function, EurekaDrive guides a language-conditioned diffusion planner with automated feedback from a VLM. Given an initial language instruction, an LLM is used to map this instruction to reward code, which then guides a pretrained diffusion policy in the environment. This rollout can be rendered into a video which is used to prompt a VLM for a revised instruction. EurekaDrive repeats this process until the goal is reached or a set number of iterations have run. 47
4.2 Visualization of EurekaDrive on a "give way" instruction goal. Each row represents an iteration of program search consisting of a language instruction, environment rollout, and VLM query. In this example, the initial instruction reward is too sparse and results in a collision. On the next iteration, the agent attempts to more slowly change lanes, but still collides with oncoming traffic in the left lane. Finally, the agent opts to simply change lanes to the right instead, avoiding any potential collisions. 48
4.3 Cumulative success rate per iteration on three complex maneuvers. Compared to using the **baseline reward** or **VLM without any feedback**, *EurekaDrive* was able to discover a larger number of successful demonstrations through sequentially updating the reward function. Although the **VLM without driving videos** eventually approached a success rate similar to *EurekaDrive*, providing video feedback enabled the framework to identify suitable reward functions more rapidly, demonstrating the importance of visual context in accelerating the search process. 52
4.4 Comparison of maneuver success rates across different planners and training settings. The results illustrate the performance of Diffusion-ES, Finetuned Diffusion-ES, Diffusion VLA, and PDM-closed under varying ES iteration budgets across six maneuver types. Finetuned Diffusion-ES achieves consistently higher success rates than the untuned baseline, while Diffusion VLA also attains comparable performance. 55

5.1 Architecture diagram for end-to-end two-stage driving policy. 61

List of Tables

2.1 Closed-loop driving results in the val14 split of [23]. Our model matches the performance of previous non-learning based planners and significantly outperforms all other models. 16
2.2 Lane following in nuPlan. 19
2.3 Diffusion-ES runtime comparison. 20

3.1 Performance on NAVSIM with standardized ResNet-34 backbone (PDMS ↑). 34
3.2 Non-grounded control task performance (Task Success × PDMS ↑). . 35
3.3 Visually-grounded control task performance (Task Success × PDMS ↑). 37

4.1 Success rate during annotated data collection. 54
4.2 Planning latency per step. Diffusion-ES latency is identical for both finetuned and non-finetuned variants. 57
4.3 Driving score comparison on nuPlan benchmark using *reduced_val14_split*. 57

Chapter 1

Introduction

A long-standing problem in autonomous driving is grappling with the long tail of rare scenarios. While modern AV approaches have seen major advances in recent years for everyday driving, the real challenge lies in tackling edge cases for which samples are risky to collect or non-existent. Especially in safety-critical situations, gracefully handling these scenarios is crucial for reliable autonomy. In this work we primarily focus on planning and policy learning, i.e., how should we learn to act when confronted with the long tail of driving scenarios?

The predominant response to this challenge has been to scale up data collection, with recent efforts yielding driving datasets spanning thousands of hours of real-world driving across diverse conditions [11, 12, 90]. However, the core issue is not the sheer amount of data but its coverage — naturalistic driving logs are dominated by routine, competent driving, and safety-critical scenarios are rare by design. No matter how much data is collected, the long tail will remain underrepresented: dangerous scenarios cannot be safely encountered on purpose, and the combinatorial complexity of real-world driving means that even large datasets will fail to cover the full distribution of scenarios a deployed system will encounter. What is needed is not more of the same data, but a way to synthetically expand coverage into precisely the regions of the distribution that naturalistic logs cannot reach.

Generative models of driving behavior offer a promising path toward this goal. Modern diffusion and flow-based trajectory models trained on large driving datasets capture rich, multimodal distributions over physically plausible driving behaviors.

Crucially, these models are not limited to reproducing behaviors present in their training data — by steering their generation with reward functions, we can synthesize counterfactual trajectories that are grounded in real scene contexts but exhibit behaviors that never occurred in the original log. This allows us to directly target the coverage gaps that naturalistic data cannot fill: rather than waiting for rare scenarios to occur, we can generate them on demand. The key challenge that remains is how to specify which counterfactual behaviors to generate — particularly for the long-tail scenarios that matter most.

This is precisely where external supervision becomes essential. In long-tail scenarios, the data manifold is by definition sparse or absent — the generative model has no reliable signal to tell it what the correct behavior should be. What is needed is external structure that exists independently of driving data density, whether from human domain experts who understand edge cases or from foundation models trained on internet-scale knowledge. This is consistent with how driving companies already approach failure triage in practice — identifying edge cases and implementing fixes is fundamentally reliant on human domain knowledge and semantic reasoning about the scene. Language is the natural medium through which this external knowledge can be expressed and made actionable — it is interpretable to humans, accessible to foundation models, and operates at a level of abstraction that captures the semantic structure of driving behavior without requiring low-level trajectory expertise. Moreover, language provides a powerful abstraction for planning and exploration — reasoning about whether to yield to an oncoming vehicle is a more natural and efficient semantic judgment than a search over low-level trajectory space, and this advantage is especially pronounced in complex, long-tail scenarios where the relevant factors are hardest to capture in low-level representations.

While language is a natural medium for expressing external knowledge about driving behavior, natural language in its unstructured form is imprecise and difficult to execute reliably — a high-level instruction like "be more aggressive" provides useful semantic guidance but is insufficient to directly steer a generative model toward a specific behavior. We focus on language programs — executable code generated from language models — as a more principled medium. Programs are precise, composable, and directly executable, allowing complex behavioral specifications to be expressed as combinations of simpler primitives. Crucially, they enable a top-down approach to

data collection: rather than gathering demonstrations and inferring labels bottom-up, we specify desired behaviors programmatically and generate demonstrations from them — automatically labeling trajectories by the semantics of the program that generated them. This means that the entire pipeline, from behavior specification to data generation to policy training, requires no human language annotation whatsoever — language programs serve as both the specification and the label, eliminating the annotation bottleneck that limits existing language-supervised approaches. This use of code as a medium for specifying behavior has seen significant traction in the broader robotics literature [52, 60], and we explore how this paradigm can be extended to autonomous driving.

We begin with Diffusion-ES, which establishes the foundation for language-programmed guidance of generative driving models. By combining a pretrained diffusion trajectory prior with evolutionary search over language-programmed reward functions, Diffusion-ES can solve challenging driving scenarios zero-shot without any language-annotated training data. The decoupling of the generative prior from the reward objective is key — rather than retraining the policy for each new task, reward programs can be swapped at test time to elicit a wide range of behaviors. However, this approach has a fundamental limitation: the policy itself learns nothing permanently. Each scenario must be solved from scratch at test time, and the approach relies on oracle language instructions being available at deployment.

This motivates GANDALF, which extends language-programmed guidance from online planning to offline data synthesis. Rather than using reward-guided inference only at test time, GANDALF runs it offline across large numbers of logged scenes to generate diverse counterfactual trajectories, which are then distilled into learned reward models and policies. The result is a policy that has internalized behaviors that previously required explicit guidance — language programs have done their work offline and are no longer needed at deployment. However, GANDALF still relies on manually specified reward programs, requiring human expertise to design reward functions that elicit the desired behaviors.

EurekaDrive closes the loop by automating the discovery of reward programs themselves. Rather than manually specifying reward functions, a vision-language model watches rollout videos and iteratively refines language programs, automatically discovering solutions to hard edge cases that the base planner cannot handle. This

shifts the locus of intelligence from human reward designers to an automated search process operating directly in program space. Crucially, finetuning on the resulting demonstrations yields policies that outperform the planner used to collect the data — suggesting that by iterating between program search, data synthesis, and policy finetuning, the framework can progressively improve without any human annotation.

The unifying claim of this thesis is that language-guided generative models can directly address the data diversity problem in autonomous driving — synthesizing the targeted counterfactual experiences that naturalistic logs cannot provide. This thesis demonstrates that this paradigm is practically viable — achieving state-of-the-art performance on established driving benchmarks, enabling zero-shot instruction following without language-annotated training data, and producing policies that outperform the planners used to collect their training data. We hope this work contributes to a growing body of evidence that the combination of generative models and language supervision is a powerful and underutilized lever for autonomous driving — a domain where the long tail is vast, data coverage is expensive to improve, and the cost of failure is high.

The rest of this thesis is organized as follows. Chapter 2 introduces Diffusion-ES, our foundational method for language-programmed test-time guidance of diffusion-based driving policies. Chapter 3 presents GANDALF, our framework for reward-guided counterfactual synthesis and distillation into learned reward models and policies. Chapter 4 introduces EurekaDrive, which automates reward program discovery via vision-language model feedback and distills the resulting demonstrations into improved driving policies. Chapter 5 presents work on latent preference finetuning, which explores semantic supervision in the form of human preference labels over high-level driving intents as a means of improving driving policies offline.

Chapter 2

Diffusion-ES: Gradient-free Planning with Diffusion for Autonomous Driving and Zero-Shot Instruction Following

In this chapter, we introduce Diffusion-ES, which establishes the foundation for reward-guided trajectory optimization that the rest of this thesis builds upon. We show how combining a pretrained diffusion model over trajectories with evolutionary search enables flexible test-time planning over arbitrary black-box reward functions, and demonstrate that language-programmed reward functions — generated via few-shot LLM prompting — enable zero-shot instruction following without any language-annotated training data. The decoupling of the generative prior from the reward objective introduced here is the key mechanism that subsequent chapters extend from online planning to offline data synthesis.

2.1 Introduction

Diffusion models have shown to excel at modeling highly complex and multimodal trajectory distributions for decision-making and control [1, 43]. Reward-gradient

guidance [43, 53, 95] has been used to test-time optimize differentiable reward functions by alternating between denoising diffusion steps and backpropagating reward gradients to the noised trajectory. In this way, sampled trajectories are pushed towards the trajectory data manifold while also maximizing the reward function at hand [43]. This decoupling of the reward function from trajectory diffusion permits a single trajectory diffusion model to be used for maximizing a variety of reward functions at test time. Reward-gradient guidance requires the reward function to be differentiable and fitted in both noisy and clean trajectories, which usually requires re-training. This limits its applicability as a general solver for trajectory optimization.

We propose Diffusion-ES , a reward-guided denoising method for optimization of non-differentiable, black-box objectives that samples and mutates trajectories using a diffusion model, guided by a reward function that operates only on the clean, final, denoised samples. Naively combining diffusion with sampling-based optimization does not work: sampling-based optimizers, like CEM [80] or MPPI [100], typically require a large population of samples across multiple iterations of selection and mutation to converge to good solutions, which, when combined with the computational cost of denoising inference, results in a prohibitively slow search process. In Diffusion-ES , *high-scoring trajectories are mutated using a truncated diffusion-denoising process*, by adding a small amount of noise and denoising them back, as shown in Figure 2.1 *right*. The amount of added noise is progressively decreased across search iterations making Diffusion-ES computationally viable.

The trajectory diffusion model used for test-time optimization in Diffusion-ES can in principle condition on any scene-relevant information to narrow the sampling to a distribution of scene-relevant trajectories. In fact, the amount of conditioning information controls a continuum between train-then-test learning and test-time planning, and a corresponding trade-off between inference speed and out-of-distribution (OOD) generalization: **1.** *The more conditioning information, the narrower the distribution to draw trajectory samples from, the faster the search.* In the extreme, no test-time reward optimization is used and our diffusion model operates as a reactive policy at test time. Indeed, many methods train diffusion policies [17] as *conditional* diffusion trajectory prediction models using rewards as conditioning information to the trajectory diffusion model [1] or finetune a diffusion policy with reinforcement learning [58, 97] or imitation learning [69]. While these methods can handle black-box

reward functions or good behaviours to imitate, we show that using them as is or with reward-guidance at test time often under-performs reward-guided denoising of an *unconditional* diffusion model, which completely decouples trajectory and reward modelling. **2.** *The less conditioning information, the wider the distribution to draw trajectory samples from, the slower the search, but the better the generalization to OOD tasks and scenarios* that require novel pairings of trajectories and scene contexts, not present in the training data. Indeed, this is the premise of test-time planning over train-then-test learning: test-time optimization of a composition of energy functions [28, 32], here, the energy of the trajectory data distribution and the energy of arbitrary reward functions, should be able to synthesize novel behaviours not seen at training time.

We show how Diffusion-ES , when combined with an unconditional diffusion model over trajectories, can achieve state-of-the-art planning performance purely through diffusion-guided black-box reward maximization. Our approach is evaluated on nuPlan [12], an established driving benchmark built on real-driving logs and estimated ground-truth perception. We achieve state-of-the-art performance for closed-loop driving, matching the performance of the previous SOTA, PDM-Closed, a sampling-based planner tailored to the nuPlan benchmark [23], as well as reactive driving policies [76, 82], deterministic or diffusion-based. Moreover, we illustrate the flexibility of Diffusion-ES by test-time optimizing language-shaped reward functions generated using few-shot LLM prompting. Using language instructions, we can solve the most challenging nuPlan scenarios, as well as synthesize entirely novel driving behaviors. We then test our model and baselines in their ability to optimize the generated reward functions to elicit the desired behaviours. Qualitative examples of behaviors generated by instruction following using our method can be found in Figure **??**. We show Diffusion-ES dramatically outperforms PDM-Closed , other sampling-based planners, as well as ablative versions of Diffusion-ES that either condition the diffusion model on the surrounding scene, or do not use any guidance at all.

In summary, our contributions are as follows:

1. We introduce Diffusion-ES , a trajectory optimization method for optimizing black-box objectives that uses a trajectory diffusion model for sampling and mutating trajectory proposals during sampling-based search. We show

Diffusion-ES matches the SOTA performance of engineered planners in closed-loop driving in nuPlan, and much outperforms them when optimizing more complex reward functions that require flexible driving behaviour, beyond lane following. To the best of our knowledge this is the first work to combine evolutionary search with diffusion models.

2. We show that Diffusion-ES can be used to follow language instructions and steer the closed-loop driving behaviour of an autonomous vehicle by optimizing the LLM-shaped reward functions, *without any training data of language and actions*. We showed that such instruction following can solve the most challenging driving scenarios in nuPlan.
3. We show extensive ablations of our model with varying amount of conditioning information which clearly reveals the trade off between inference speed and out-of-distribution (OOD) generalization in driving.

2.2 Related work

Diffusion models for decision-making and trajectory optimization Diffusion models [35, 67, 88, 89] learn to approximate the data distribution through an iterative denoising process and have shown impressive results on image generation [25, 75, 79, 81]. They have been used for imitation learning for manipulation tasks [17, 69, 77, 101], for controllable vehicle motion generation [13, 44, 109] and for video generation of manipulation tasks [29, 105]. Works of [44, 109] use diffusion models to forecast offline vehicle trajectories. To the best of our knowledge, this is the first work to use diffusion models in closed-loop driving.

Learning versus planning for autonomous driving Learning to drive from imitating driving demonstrations is prevalent in the research and development of autonomous vehicles [5, 7, 15, 16, 20, 49, 71, 72, 76, 108]. Many preeminent imitation methods assume the underlying action distribution is unimodal, which is problematic when training from multimodal expert demonstrations. Objectives and architectures that can better handle multimodal trajectory prediction have been proposed [22, 31, 47, 66, 84, 93]. We show that diffusion models are well-suited for driving and can be used to synthesize rich complex behaviors from multimodal demonstrations.

On the other hand, conventional autonomy stacks do not rely on learning at all for decision making, and rather rely on optimizing manually engineered cost functions online [30, 64, 113]. Recently, PDM-Closed [23] achieved state-of-the-art performance on the nuPlan driving benchmark by purely relying on test-time planning and heuristics for selecting trajectory proposals. Other prior works aim to incorporate the benefits of offline learning for test-time planning by performing sampling-based planning over learned cost maps [107] or doing gradient-based optimization over learned dynamics models [78]. We extend this line of work by showing how diffusion-based generative models can be combined with sampling-based planning.

Language-conditioned policies for autonomous driving Recently there has been significant progress made towards language-conditioned policies for driving. GAIA-1 [36] is a generative world model capable of multimodal video generation that leverages video, language and actions to synthesize driving scenarios which can comply with given language instructions. However, GAIA-1 does not execute any actual control inputs.

LLMs trained from Internet-scale text have shown impressive zero-shot reasoning capabilities for a variety of downstream language tasks when prompted appropriately, without any weight fine-tuning [10, 57, 98]. Recent works have shown that LLMs can be prompted to map language instructions to language subgoals [38, 39, 103, 111] action programs [33, 52, 91] or cost maps [40] with appropriate plan-like or program-like prompts. Our work follows few-shot prompting of LLMs to shape driving reward functions. We extend previous methods by using Python generators to produce reward functions which maintain internal state across calls. Works of [61, 99, 104] use LLMs to predict low-level control signals given high-level scene descriptions and language instructions. However, none of these approaches evaluate their performance on closed-loop driving, which is significantly more difficult than open-loop trajectory forecasting [23]. [83] is similar to us, but only considers a highly simplified driving setup and does not report results on a standardized benchmark with strong baselines.

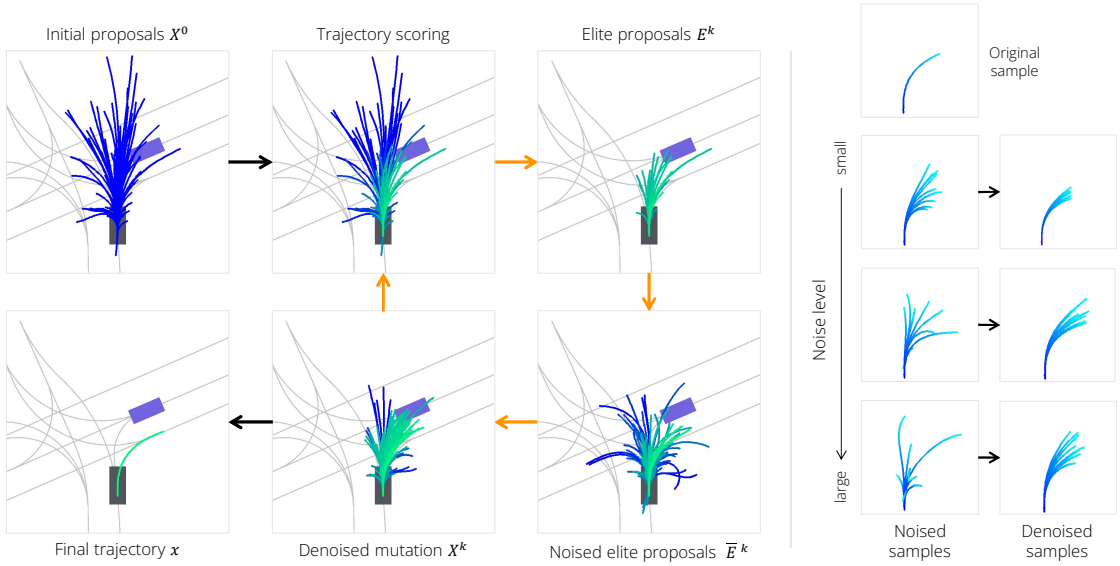


Figure 2.1: Overview for Diffusion-ES . *Left:* Generative evolutionary search with Diffusion-ES . Trajectories are colored by rewards (blue is low, green is high). *Right:* We visualize the mutations for varying noise levels. Color denotes timestep along trajectory. While noise perturbations alone can lead to unrealistic trajectories, denoising helps project samples back onto the trajectory data manifold.

2.3 Method

2.3.1 Background

Diffusion models A diffusion model captures the probability distribution $p(x)$ through the inversion of a forward diffusion process, that gradually adds Gaussian noise to the intermediate distribution of a initial sample x. The amounts of added noise depend on a predefined variance schedule $\beta_t \in (0,1)_{t=1}^{T}$, where T denotes the total number of diffusion timesteps. At diffusion timestep t, the forward diffusion process adds noise into x using the formula $x_t = \sqrt{\bar{\alpha}_t}x + \sqrt{1-\bar{\alpha}_t}\epsilon$, where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ is a sample from a Gaussian distribution with the same dimensionality as x. Here, $\alpha_t = 1 - \beta_t$, and $\bar{\alpha}t = \prod i = 1^t \alpha_i$. For denoising, a neural network $\hat{\epsilon} = \epsilon_\theta(x_t; t)$ takes input as the noisy sample x_t and the diffusion timestep t, and learns to predict the added noise ϵ. To generate a sample from the learned distribution $p_\theta(x)$, we start by drawing a sample from the prior distribution $x_T \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$ and iteratively denoise

this sample T times with ϵ_θ. The application of ϵ_θ depends on a specified sampling schedule [35, 89], which terminates with x_0 sampled from $p_\theta(x)$. Diffusion models can be easily extended to model $p(x|\mathbf{c})$, where $\mathbf{c}$ is some conditioning signal, such as the expected future rewards , by adding an additional input to the denoising neural network ϵ_θ.

Evolutionary strategies Evolutionary strategies (ES) are a family of population-based *gradient-free* optimization algorithms which can maximize arbitrary black-box reward functions $R(x) : \mathbb{R}^d \to \mathbb{R}$ without any training, where x is the variable we optimize over. ES iteratively update a search distribution $q(x)$ to maximize expected rewards $\mathbb{E}_{x\sim p_\theta(x)}[R(x)]$. While distribution-based ES approaches such as CEM and CMA-ES represent $q(x)$ explicitly (often as a unimodal Gaussian), we can also represent $q(x)$ non-parametrically as a set of high-performing solutions without making strong assumptions about the functional form of q.

2.3.2 Diffusion-ES

Diffusion-ES is a trajectory optimization method that leverages gradient-free evolutionary search to perform reward-guided sampling from trained diffusion models, for any black-box reward function $R(x)$. Specifically, we use a trained diffusion model ϵ_θ to initialize the sample population and we use a truncated diffuse-denoise process to mutate samples while staying in the data manifold. The control flow of Diffusion-ES is shown in Figure 2.1 (left).

Initializing the population with diffusion sampling We begin by sampling an initial population X^0 of M trajectory samples using our diffusion model:

$$X^0 = \{x_i\}_{i=1}^M \sim p_\theta(x), \tag{2.1}$$

where X^k is the population at iteration k. This involves a complete pass through the reverse diffusion process. Given we use an unconditional diffusion model, these samples are scene agnostic and can always be used without re-sampling them at each timestep. We can also modify the initial population by including samples generated by other approaches or mixing in solutions from the previous timestep to warm-start our optimization.

Sample scoring At each iteration k, we score the samples in our population

$\{R(x_i)|x_i \in X^k\}_{i=1}^M$. Note that our population consists of "clean" samples so we do not need a reward function which can handle "noisy" samples, which gives us significant flexibility compared to guidance methods that perform classifier-based guidance.

Selection We use rewards to decide which samples we should select to propagate to the next iteration. Similar to MPPI [100], we resample X^{k+1} as follows:

$$q(x) = \frac{\exp(\tau R(x))}{\sum_{i=1}^M \exp(\tau R(x_i))} \tag{2.2}$$

$$E^{k+1} = \{x_i \overset{\text{iid}}{\sim} q(x)\}_{i=1}^M, \tag{2.3}$$

where E^{k+1} represents our elite set which is kept from iteration k, and τ is a tunable temperature parameter controlling the sharpness of q.

Mutation using truncated diffusion-denoising We apply randomized mutations to E^{k+1} for exploration. Prior evolutionary search methods resort to naive Gaussian perturbations which do not exploit any prior knowledge about the data manifold. **Our key insight is to leverage a truncated diffusion-denoising process to mutate trajectories in a way the resulting mutations are part of the data manifold.** We can run the first t steps of the forward diffusion process to get noised elite samples $\bar{E}^{k+1}$:

$$\bar{E}^{k+1} = \{\sqrt{\bar{\alpha}_N}x + \sqrt{1-\bar{\alpha}_N}\epsilon | x \in E^{k+1}\}, \tag{2.4}$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{1})$. Then we can run the last t steps of the reverse diffusion process to denoise the samples again, giving us clean samples X^{k+1}:

$$X^{k+1} = \{x \sim p_\theta(x|\bar{x}) | \bar{x} \in \bar{E}^{k+1}\}. \tag{2.5}$$

In practice, the number of timesteps t of the truncated diffusion process is a tunable time-dependent hyperparameter t_k which controls the mutation strength at each iteration k. This is visualized in Figure 2.1 (right). We find that linearly decaying the number of mutation diffusion steps t_k from 5 to 1 over 20 search steps works best in our experiments.

2.3.3 Mapping language instructions to reward functions with LLM prompting

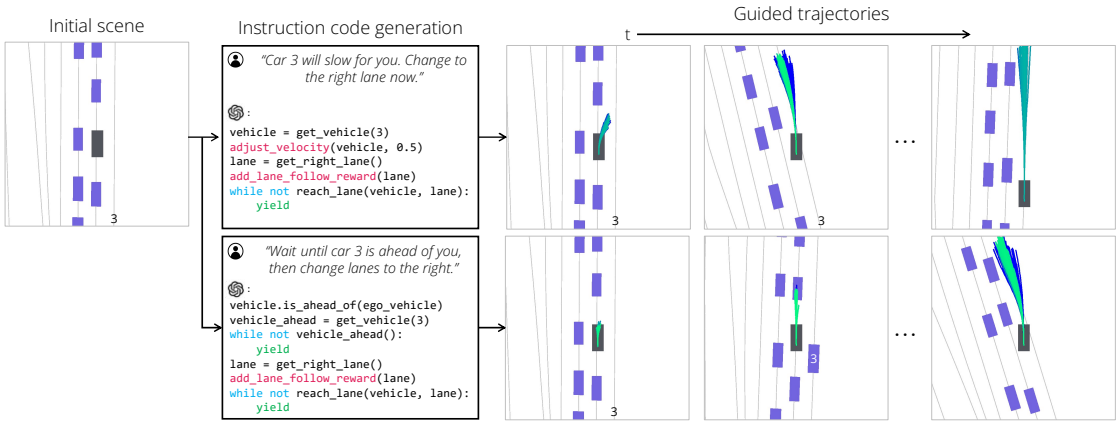


Figure 2.2: Shaping reward functions with language instructions using LLM prompting and optimizing them with Diffusion-ES . For the same initial scene, we show two distinct language instructions, along with their generated reward shaping code and generated trajectories from Diffusion-ES .

To follow driving instructions given in natural language, we map them to black-box reward functions which we optimize with Diffusion-ES . We adopt a similar approach to [46, 106] which uses LLMs to synthesize reward functions from language instructions. Reward functions are composable and allow us to seamlessly combine language guidance with other constraints. This is crucial in driving where we constantly optimize many different objectives at once (e.g., safety, driver comfort, route adherence).

Similar to prior work, we expose a Python API which can be used to extract information about entities in the road scene. Since many of the basic reward signals in driving do not change from scenario to scenario (e.g., collision avoidance or drivable area compliance), we allow the LLM to write *reward shaping* code which modifies the behavior of the base reward function, as opposed to generating everything from scratch. Reward shaping can add auxiliary reward terms (e.g., a dense lane-reaching reward) or re-weight existing reward terms. We show generated code examples in Figure 2.2.

Our goal is to handle general and complex language instructions with temporal

dependencies, such as *"Change lanes to the left, then pass the car on the right, then take the exit"*. Previous works [40, 106] produce a stationary reward function, i.e., one which is fixed during planning. This can make it challenging to express sequential plans solely through rewards. We find that a much more natural and succinct way of capturing these plans in code is through the use of *generator functions* which retain internal state between calls. All our prompts and code examples can be found in the supplementary file. In Section 2.4, we show how optimizing these language-shaped reward functions can synthesize rich and complex driving behaviors that comply with the language instructions.

2.4 Experiments

We first evaluate Diffusion-ES on closed-loop driving in nuPlan [12], an established benchmark that uses estimated perception for vehicles, pedestrians, lanes and traffic signs. Our model and baselines are evaluated on their ability to drive safely and efficiently while having access to close-to-ground-truth perception output provided by the dataset. We also consider a suite of driving instruction following tasks. We map instructions to shaped reward functions with LLM prompting, and evaluate Diffusion-ES and baselines in their ability to optimize the generated reward functions and accurately follow the instructions. Our experiments aim to answer the following questions:

1. How does Diffusion-ES compare to existing sampling-based planners and reward-gradient guidance for trajectory optimization?
2. How does Diffusion-ES compare to SOTA reactive driving policies that directly map environments' state to vehicle trajectories?
3. Can the hardest nuPlan driving scenarios be solved by assuming access to a human teacher giving instructions in natural language, without any additional training data?
4. Does scene conditioning for the diffusion model benefit Diffusion-ES ?

2.4.1 Closed-loop driving

We evaluate our model on the nuPlan Val14 planning benchmark [23]. We specifically consider the reactive agent track of the nuPlan benchmark since it is the most difficult and realistic of the evaluation settings in nuPlan.

Model setup and hyperparameters For Diffusion-ES , we train a diffusion model over ego-vehicle trajectories consisting of 2D poses (x, y, θ) predicted 8 seconds into the future at 2Hz, leading to an overall action dimension of 48. Unlike prior work [43], we model the distribution over actions only rather than modeling states and actions. We use a population size of $M = 128$ in our experiments. Our diffusion model is trained with $T = 100$ denoising steps.

Reward function We adopt a modified version of the scoring function used in PDM-Closed [23] as our reward function. To compute rewards, we convert our predicted trajectory to low-level control inputs using an LQR tracker. These control inputs are fed into a kinematic bicycle model [70] which propagates the dynamics of the ego-vehicle. We follow [23] and forecast the motion of other agents by assuming constant velocity. These simulated rollouts are then scored following the nuPlan benchmark evaluation metrics. We also add auxiliary reward terms to penalize proximity to the leading agent and enforce speed limits.

Note that this reward function is *not differentiable* due to the tracker and the use of non-differentiable heuristics for assessing traffic violations. Additionally, training a model to regress rewards is challenging since the nuPlan dataset contains no instances of serious traffic infractions.

Evaluation metric We report our results using **driving score**, which aggregates multiple planning metrics related to traffic rule compliance, safety, route progress, and rider comfort. This is the standard evaluation metric used in nuPlan.

Baselines We consider the following baselines:

- *UrbanDriverOL* [82], a deterministic transformer policy trained with behaviour cloning and augmentations. Unlike in [82], the nuPlan implementation does not perform closed-loop training.

	Method	Driving Score (↑)
Train-then-test	UrbanDriverOL [82]	65
	PlanCNN [76]	72
	Diffusion policy	50
Test-time plan	IDM [80]	77
	PDM-Closed [23]	**92**
	Diffusion-ES (ours)	**92**

Table 2.1: Closed-loop driving results in the val14 split of [23]. Our model matches the performance of previous non-learning based planners and significantly outperforms all other models.

- *PlanCNN* [76] a deterministic imitation policy which encodes a rasterized BEV map using a CNN backbone.
- *IDM* [34]: a heuristic rule-based planner which adjusts its speed to maintain a safe distance to the leading vehicle. It is also used to control the behaviour of agents in nuPlan.
- *PDM-Closed* [23]: an MPC-based planner which generates path proposals using lane centerlines, and rolls out trajectories similarly to us. Instead of iteratively optimizing rewards, PDM-Closed simply executes the highest-performing proposal after one round of scoring. It is the current state-of-the-art on the nuPlan Val14 benchmark.
- *Diffusion Policy*: a diffusion model we consider that conditions on scene features to predict a vehicle trajectory directly. We encode scene features using the transformer feature backbone from Urban Driver [82]. We train it with imitation learning and augmentations. This is similar to the unconditional trajectory model in Diffusion-ES with additional conditioning on scene features.

We show quantitative results in Table 2.1. We draw the following conclusions:

1. Diffusion-ES matches the prior state-of-the-art, PDM-Closed and substantially outperforms all other baselines. PDM-Closed is a sampling-based planner that relies on domain-specific heuristics to generate trajectory proposals whereas Diffusion-ES learns these proposals from data. Both use a similar reward function and dynamics model for the agents in the scene.

2. There is a large gap in performance between reactive neural policies and test-time planners, also pointed out in recent work [23]. We hypothesize that this is because compared to other control benchmarks, nuPlan has a much richer observation space as scenes are densely populated by dynamic actors, many of which are irrelevant to the ego-agent. This can make it challenging for learning-based methods to generalize, which motivates the need of test-time optimization. We attribute this gap primarily to the limited scale and behavioral diversity of nuPlan rather than a fundamental limitation of diffusion policies, which have demonstrated strong results in other control settings [17]. Test-time search is a practical solution in this data regime, but the more important implication is that closing this gap requires richer and more diverse training data. This observation directly motivates the offline data synthesis approaches in Chapters 3 and 4, which use the same reward-guided generation mechanism to produce diverse counterfactual demonstrations at scale—addressing the root cause rather than compensating for it at inference time.

3. Diffusion-ES substantially outperforms diffusion policy. Qualitatively, our diffusion policy has a tendency to randomly change lanes, which causes the ego-vehicle to reach out-of-distribution scenarios faster. Diffusion-ES leverages the expressiveness of generative modeling while using test-time optimization to improve generalization.

4. Diffusion-ES matches but does not substantially outperform PDM-Closed. Both methods achieve extremely high driving scores on this benchmark, and as noted in [23], a non-trivial fraction of the remaining failures are unsolvable scenarios (e.g., unavoidable collisions caused by other agents) which place a ceiling on achievable performance. Moreover, the nuPlan Val14 benchmark heavily favors conservative driving: lane changes are rarely required and assertive maneuvers are not rewarded, meaning a relatively limited set of trajectory proposals — such as PDM-Closed's centerline-based paths — is sufficient to perform well. This suggests that the expressiveness of a learned diffusion prior is not fully exercised on this benchmark. However, as we show in the following section, when evaluating on more challenging language-guided tasks that require diverse and assertive behaviors, the gap between Diffusion-ES and PDM-Closed widens considerably.

2.4.2 Language instruction following

One drawback of the nuPlan driving benchmark is that encourages highly conservative driving behaviors. For example, PDM-Closed [23] holds the state-of-the-art in the nuPlan Val14 benchmark while being unable to change lanes, since its path proposals only consider the lane the ego-vehicle is currently on. However, lane changing is not necessary for good driving performance in the current nuPlan benchmark.

To evaluate Diffusion-ES and baselines in their ability to optimize arbitrary reward functions, we consider eight language instruction following tasks, each taken from an existing driving log in the nuPlan benchmark. In each task, the language instruction requires the ego-vehicle to perform a specific driving maneuver that solves a challenging driving scenario. In most scenarios there will be no examples of the instructed behavior anywhere in nuPlan. For instance, the lane weaving task requires the ego-vehicle to aggressively change multiple lanes in dense urban traffic. Task descriptions, language instructions, and prompts are provided in the appendix. We use the method described in 2.3.3 to generate executable Python code given a language instruction that adapts the initial reward function of Section 2.4.1, giving us a language-shaped reward function for each scenario. Our model and baselines will optimize the same language-shaped reward function.

Evaluation metric We evaluate our model and baselines using **task success rate**, which measures how frequently the agent was able to successfully complete the designated task. To increase the difficulty of the tasks, we randomize the behavior of other vehicle agents by adding noise to their IDM parameters at sporadic intervals during each episode. All scores reported are averaged across ten random seeds.

Baselines We compare to the following baselines:

- *PDM-Closed*: this is adapted to this setting by using our language-shaped reward function in place of the original reward function.
- *PDM-Closed-Multilane*: a modified variant of PDM-Closed which considers a wider range of laterally offset paths, allowing for lane changes.
- *Conditional Diffusion-ES* : Diffusion-ES that uses a conditional diffusion model instead of an unconditional one.

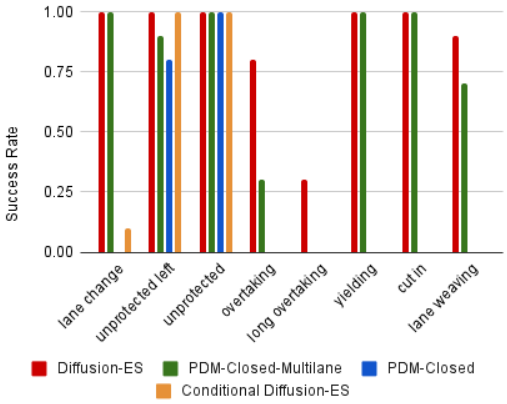


Figure 2.3: Following driving instructions. Diffusion-ES outperforms all baselines in optimizing complex language-shaped reward functions.

Figure 2.3 shows the success rates on the controllability tasks. We draw the following conclusions:

1. Diffusion-ES outperforms all baselines. Although PDM-Closed-Multilane has substantially improved performance compared to PDM-Closed due to more diverse trajectory proposals, it is still weaker than Diffusion-ES on 4 out of 8 tasks. This highlights the weakness of relying on handcrafted rules for proposal generation.

2. Diffusion-ES performs significantly worse with a conditional diffusion model. The conditional diffusion model is much harder to guide since the scene context causes fewer samples to be in-distribution. This effect is particularly pronounced in the language controllability setting, where the reward functions explicitly request behaviors that deviate substantially from the training distribution (e.g., aggressive lane changes, sudden stops). In these cases, the scene-conditioned model has learned a strong prior over typical behaviors for a given scene context, and reward-guided search must fight against this prior to produce the desired out-of-distribution behavior. The unconditional model, by contrast, provides a weaker but more flexible prior that is easier to steer toward novel behaviors. We note that scene conditioning is not inherently detrimental to reward-guided planning — in Chapters 3 and 4, we successfully use scene-conditioned priors for counterfactual trajectory generation. The key difference is that those settings pair scene conditioning with substantially more diverse training data, which broadens the model's support and makes guided

Method	Lane Error	Speed Error
CEM	2.34	2.05
MPPI	3.29	2.74
Reward-gradient guidance	1.22	0.96
Diffusion-ES (ours)	**0.61**	**0.79**

Table 2.2: Lane following in nuPlan.

generation less likely to leave the distribution.

2.4.3 Lane following

To compare Diffusion-ES against reward-gradient guidance, we consider a simplified lane following task with a differentiable reward function, which consists of two terms: one penalizing lateral deviation from the lane (**lane error**), and one penalizing deviation from a target speed (**speed error**). We sample 14 scenarios, one of each scenario type in nuPlan, and report average planning costs across all scenarios.

Baselines We consider the following baselines:

- *CEM* [80]: a widely used ES method which parameterizes the search distribution q as a Gaussian. CEM iterates between sampling from q and fitting q to the best samples.
- *MPPI* [100]: similar to CEM but rather than keep a fixed number of elites, MPPI samples proportional to rewards.
- *Reward-gradient guidance* [43]: we directly optimize the ground-truth planning objective with gradient descent during the denoising process.

We show quantitative results in Table 2.2. We draw the following conclusions: **1.Diffusion-ES outperforms the differentiable reward-gradient guidance baseline *even though the objective is differentiable***. We hypothesize that this is because although the ground truth reward function is available, it may not provide suitable guidance for intermediate noisy trajectories. This highlights a key advantage of our method over prior work, which is that we can optimize novel objectives without needing to train a reward regressor on noisy samples. **2. Both diffusion-based methods significantly outperform sampling-based planners that do not**

Method	Wallclock time (s)
Diffusion	1.11 ± 0.02
Diffusion-ES	5.85 ± 0.11
Diffusion-ES (*optimized*)	0.50 ± 0.01

Table 2.3: Diffusion-ES runtime comparison.

leverage diffusion. This is consistent with our hypothesis that diffusion guidance can optimize trajectories much more efficiently than conventional ES methods. Videos of our method driving in all experimental settings can be found in our project page diffusion-es.github.io.

2.4.4 Runtime analysis

Diffusion-ES can be used in real-time with some minor optimizations. By using fewer diffusion steps $T = 10$, smaller population size $M = 32$ and less iterations $K = 2$, our method can be run at the same frequency as the simulator (2 Hz) at a small cost to performance (nuPlan driving score drops from 92 to 91). We report the average wallclock time for inference at every timestep over 100 trials in Table 2.3.

2.5 Conclusion

We presented Diffusion-ES , a method for black-box reward guided diffusion sampling. We showed that Diffusion-ES can effectively optimize reward functions in nuPlan for driving and instruction following, and outperforms engineered sampling-based planners, reactive deterministic or diffusion policies, as well as differentiable reward-gradient guidance. We showed how our method can be used to follow language instructions without any language-action trajectory data, simply using LLM prompting to generate shaped reward maps for test-time optimization. Our experiments show the trade-off between inference speed and OOD generalization during scene conditioning in diffusion policies.

A key limitation of this approach is that the policy itself learns nothing permanently — each scenario must be solved from scratch at test time via expensive iterative search, and the approach assumes that oracle language instructions are available at

deployment. In the next chapter, we show how the same reward-guided inference mechanism can instead be run offline over large numbers of logged scenes to generate diverse counterfactual trajectories, which are then distilled into learned policies that no longer require test-time guidance.

Chapter 3

GANDALF: Language-Controllable Driving without Labels via Diverse Reward-Guided Synthesis

3.1 Introduction

Diffusion-ES demonstrated that reward-guided inference over a generative trajectory prior can solve challenging driving scenarios at test time. However, this approach has a fundamental limitation: the policy itself learns nothing permanently — each scenario must be solved from scratch, and oracle language instructions must be available at deployment. A natural question is whether the same mechanism can be used offline to generate diverse training data at scale, distilling the benefits of reward-guided planning into learned policies that no longer require explicit guidance.

The key challenge is behavioral diversity. Naturalistic driving logs are dominated by routine driving, with safety-critical events and rare maneuvers underrepresented by orders of magnitude. Reward models require diverse and contrastive examples to distinguish good from bad behavior, and language-controllable policies need counterfactual trajectories that each logged scene's single actualized trajectory cannot provide. Reward specification offers a natural lever — by varying reward functions, we can elicit a wide range of behaviors without modifying the underlying model — but

generating these trajectories typically requires expensive real-world collection or high-fidelity sensor resimulation. We sidestep this by scoring counterfactual trajectories via non-reactive simulation against offline logs, which is inexpensive and requires no new sensor data, allowing us to massively scale up behavioral diversity.

We address this with GANDALF (**G**enerating **A**uto**n**omy **D**emonstrations via **A**nalytic and **L**earned Reward **F**unctions), a framework for generating diverse driving demonstrations via reward-guided planning over a scene-conditioned flow prior. By building on Diffusion-ES, which supports black-box reward functions, GANDALF can incorporate arbitrary reward specifications at inference time without retraining, in contrast to rule-based planners which can be brittle and require careful tuning when reward functions change. Critically, separating the generative prior from the reward objective resolves the tension between diversity and quality inherent in using a generative model directly as a policy: we show experimentally that increasing the diversity of the prior does not degrade output quality, because reward-guided inference selects high-quality trajectories from the population.

We first apply GANDALF to improve reward-based planning in perception settings. Online planning with reward functions has proven highly effective in state-based benchmarks like NuPlan, where ground-truth state makes reward computation reliable. This success has not transferred to perception-based settings, where analytic rewards are unreliable due to compounding perception errors, and learning reward models directly is difficult without diverse training data. GANDALF addresses this directly: by generating reward-labeled counterfactual trajectories at scale over logged scenes, we distill these into observation-conditioned reward models that require no privileged state at deployment. We evaluate on NAVSIM, where learned rewards can outperform analytic rewards in perception-based settings, achieving state-of-the-art performance.

We then show the same framework extends naturally to language controllability. A key challenge for language-conditioned driving policies is that training requires diverse behaviors and language labels collected jointly, which is expensive. GANDALF sidesteps this entirely: by varying reward functions over logged scenes, we generate task-diverse trajectories spanning a wide range of driving styles and objectives. Because the reward functions used to generate each trajectory encode its semantics, we can label trajectories with language descriptions programmatically, producing paired vision-language-action data without any human annotation. We train a

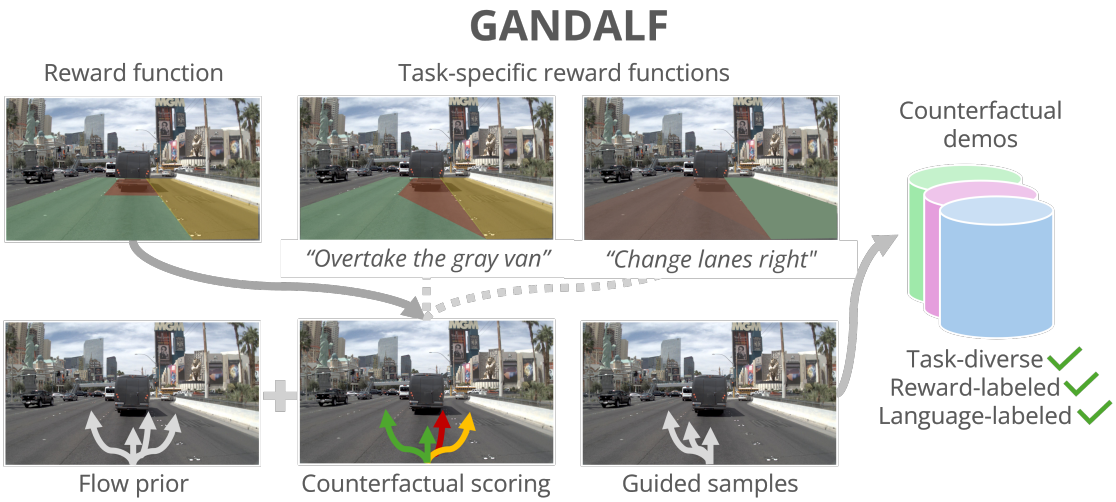


Figure 3.1: GANDALF generates diverse driving demonstrations by guiding a scene-conditioned flow prior with reward functions over logged scenes. By varying the reward specification—general driving quality, task-specific objectives (e.g., "change lanes right"), or visually-grounded instructions (e.g., "overtake the gray van")—we produce counterfactual trajectories that are task-diverse, reward-labeled, and language-labeled, without new data collection or human annotation.

VLM-conditioned reward model on these trajectories and show that it nearly matches dedicated per-task reward modules, while outperforming modular pipelines on visually-grounded tasks by avoiding error propagation from multi-stage approaches.

We make the following contributions:

- We propose GANDALF, a framework for generating diverse counterfactual driving demonstrations via reward-guided planning over a scene-conditioned flow prior, without requiring sensor resimulation.
- We show that distilling reward-labeled trajectories into observation-conditioned reward models enables learned rewards to outperform analytic rewards in perception-based settings, achieving state-of-the-art performance on NAVSIM.
- We demonstrate that the same framework extends to language controllability: by varying reward functions and labeling trajectories programmatically from reward semantics, we train VLM-conditioned reward models that follow visually-grounded instructions without any human-annotated language data.

3.2 Related work

Diffusion and flow models for planning. Diffusion models have emerged as powerful tools for trajectory generation and planning, capturing multimodal distributions over futures. Diffuser [43] first applied diffusion models to offline RL, planning by iteratively denoising trajectories. Subsequent work has extended this to goal-conditioned planning [2], hierarchical decision-making [53], and real-time control via diffusion policies [17]. For autonomous driving, trajectory diffusion models have been used for motion forecasting [45] and planning [109]. Most relevant to our work, Diffusion-ES (Chapter 2) introduced gradient-free reward optimization over diffusion models via evolutionary search, enabling black-box reward guidance without retraining. GANDALF builds on this inference procedure but extends its application: beyond online planning, we use reward-guided generation to synthesize training data at scale, closing a loop between generation and learning.

Learning-based motion planning for driving. Autonomous driving planners span a spectrum from modular pipelines [12] to end-to-end learned policies [20]. Modular approaches decompose the problem into perception, prediction, and planning, offering interpretability but suffering from error accumulation. End-to-end methods learn direct mappings from sensors to controls [7], but tie behavior to the training distribution and objectives. Recent work has sought middle grounds: incorporating learned components into structured planners [23], or using world models for planning [36]. Our approach is complementary—we learn a trajectory prior that captures feasible behaviors, then modulate objectives at inference time via rewards, combining the flexibility of learned models with the controllability of reward-based planning.

Reward learning and specification. Specifying reward functions for complex tasks remains challenging. Inverse reinforcement learning [112] and preference learning [19] infer rewards from demonstrations, while recent work uses language models to generate reward code from natural language [106]. Diffusion-ES demonstrated that LLMs can map instructions to reward programs for driving. We extend this direction by training learned reward models that distill analytical rewards into observation-conditioned predictors, removing the need for privileged state at test time. Our

VLM-conditioned reward module further enables language-specified objectives with visual grounding.

Data augmentation and counterfactual generation. Data scarcity and distribution shift are persistent challenges in learning-based driving. Prior work has addressed this through simulation [27], domain randomization [94], and generative augmentation of sensor data [48]. Rather than augmenting observations, we augment trajectories: using reward-guided inference to generate counterfactual behaviors over real logged scenes. This produces diverse demonstrations—including rare maneuvers underrepresented in naturalistic data—while remaining grounded in real perception inputs. The approach shares motivation with hindsight relabeling [3] and goal-conditioned data augmentation [65], but operates at the trajectory level via explicit reward specification.

Vision-language-action models. Vision-language models have been adapted for robotic control by conditioning policies on language instructions [9]. Training such models typically requires paired language-action data, which is expensive to collect at scale. Prior work has used language annotations from humans [42], automatic labeling via captioning models [63], or hindsight instruction relabeling [59]. We sidestep the annotation bottleneck entirely: since we control the reward function that generates each trajectory, we can programmatically assign language labels based on the generating task, producing diverse vision-language-action data without human annotation.

3.3 Scene-Conditioned Flow Prior

3.3.1 Overview

Let $o = (o_{-H}, \ldots, o_0)$ denote a history of sensor observations (camera and lidar) ending at the current timestep, and let $\tau = (x_1, \ldots, x_T) \in \mathbb{R}^{T\times 2}$ denote a future ego trajectory as a sequence of 2D positions at fixed time intervals. We learn a generative model $p_\theta(\tau \mid o)$ over trajectories conditioned on the observation history, and treat it not as a policy but as a prior. This prior captures the distribution of scene-consistent

futures—encoding what behaviors are feasible given the observed context—without committing to any particular driving objective.

This separation of dynamics from objectives is central to our approach. Behavior emerges through reward-guided inference: given a reward function $r(\tau)$ encoding task-specific preferences, we sample trajectories from the prior and select those scoring highly under r. Because the reward enters only at inference time, the same prior supports online planning under different objectives without retraining, as well as offline generation of counterfactual trajectories by varying rewards over logged scenes.

3.3.2 Model Architecture

Conditional flow matching. We parameterize p_θ as a conditional flow matching model, training a velocity field $v_\theta(\tau^{(t)}, t \mid o)$ that transports samples from a Gaussian base distribution to the data distribution over a continuous time index $t \in [0, 1]$. We use a linear-Gaussian probability path, where $\tau^{(t)} = (1-t)\epsilon + t\tau$ for noise $\epsilon \sim \mathcal{N}(0, I)$ and data τ. At inference, trajectories are generated by integrating the velocity field.

Perception backbone. The observation history consists of multi-camera RGB images, LiDAR, and ego state. An encoder f_ϕ maps observations to scene, agent, and map tokens: $(\mathbf{z}_{\text{scene}}, \mathbf{z}_{\text{agent}}, \mathbf{z}_{\text{map}}) = f_\phi(o)$. Agent and map tokens are supervised via auxiliary object detection and BEV semantic map prediction tasks respectively. These auxiliary objectives ground the token representations in scene structure, providing the flow decoder with semantically meaningful features.

Trajectory decoder. Given a noisy trajectory $\tau^{(t)}$ at flow time t, the trajectory decoder cross-attends to $\mathbf{z}_{\text{scene}}$, $\mathbf{z}_{\text{agent}}$, and $\mathbf{z}_{\text{map}}$ to predict the velocity field $v_\theta(\tau^{(t)}, t \mid o)$. Cross-attention is spatially-aware: we use rotary embeddings encoding 2D positions to provide relative positional information between trajectory waypoints and scene tokens. A separate trajectory flow decoder generates forecasts for non-ego agents in their respective frames, which we use for downstream planning.

3.3.3 Reward-Guided Inference

Flow guidance with evolutionary search. Given a reward function $r(\tau)$, we perform online planning following Diffusion-ES (Chapter 2). We initialize a population of N candidate trajectories by sampling from the prior $\{\tau_i\}_{i=1}^{N} \sim p_\theta(\tau \mid o)$, then iteratively refine this population over K iterations. At each iteration, trajectories are scored under r, elite trajectories are selected based on their rewards, and these elites are mutated using a truncated diffusion process: a small number of forward noising steps followed by denoising. This perturbation mechanism allows efficient exploration while keeping trajectories on the learned data manifold.

Reward specification. In this work, we consider two types of reward function $r(\tau)$: analytical or learned. Analytical reward functions are manually designed functions that score trajectories using scene state—agent positions, lane geometry, traffic signals—and encode interpretable objectives such as collision avoidance or route progress. They offer transparency and precise control, but require access to structured state representations, either from ground-truth annotations offline or perception outputs online. Alternatively, learned reward modules can be trained to predict rewards directly from observations, removing the dependence on explicit state estimation at test time. They can capture objectives that are difficult to specify analytically, but require supervision and may be less interpretable. We show that these approaches are complementary: analytical rewards can generate training signal for learned reward models, which in turn enable deployment without privileged state access.

3.4 Counterfactual Synthesis and Distillation

Reward-guided inference over the scene-conditioned flow prior enables controllable synthesis of trajectory demonstrations: by modulating the choice of reward function, we can generate trajectories exhibiting diverse behaviors across logged scenes. These synthetic demonstrations serve as training data for downstream models—learned reward modules, fine-tuned policies, and vision-language-action models. We organize this section into two parts: first, training learned reward models to improve general

driving performance; second, generating task-diverse data via reward specification and distilling it into task-specific modules and policies.

3.4.1 General Driving

Expert demonstrations consist almost entirely of high-quality behavior. A reward model trained only on expert data lacks the contrastive examples needed to discriminate between good and poor trajectories. We address this by generating diverse demonstrations from the flow prior and scoring them with an analytical reward, producing a dataset with broad coverage across trajectory quality.

Rewards from offline logs. We use a manually designed reward inspired by PDMS [24] penalizing collisions, lane departures, route deviation, and harsh dynamics. This requires access to scene state such as agent trajectories, lane geometry, and traffic signals. In offline logs, this state is available as ground-truth annotation—and crucially, we have perfect knowledge of how the scene unfolds, allowing exact collision checking against recorded agent futures.

Data generation. For each scene in the training set, we run Diffusion-ES using this analytical reward. Crucially, we retain and score *all* candidate trajectories across all iterations of the evolutionary process—not just the final elites. Because Diffusion-ES iteratively scores, selects, and resamples trajectories, intermediate populations naturally span a wide range of quality: early iterations contain diverse but often low-scoring candidates, while later iterations concentrate around high-reward regions. Scoring each candidate against the ground-truth analytical reward produces a richly contrastive dataset, with trajectory-reward pairs ranging from collisions and lane departures to near-optimal driving, all grounded in the flow prior's learned manifold.

Learned reward model. We train a learned reward model $r_\psi(\tau \mid o)$ to regress to the analytical scores. The model architecture is derived from the trajectory decoder. Learnable reward query tokens are concatenated to the input trajectory tokens and decoded to scalar reward values. Since our analytic reward consists of multiple components corresponding to different metrics, each one corresponds to its own query

token and a final scalar reward is computed by composing the terms. At test time, this learned reward can replace the analytical one, enabling reward-guided inference without requiring ground-truth state.

3.4.2 Controllability

Beyond general driving, the same framework enables generating trajectories for specific maneuvers—lane changes, speed variations, vehicle following—by swapping the reward function. Since we control which reward generates each trajectory, we can programmatically annotate the resulting data with language descriptions, producing paired vision-language-action data without human annotation.

Task-specific rewards. We define analytical rewards for seven tasks: stay in lane, change lane left, change lane right, accelerate, decelerate, follow vehicle, and overtake vehicle. Each modifies or extends the general driving reward to encourage the target behavior—for instance, the lane change reward adds a term incentivizing lateral displacement toward the adjacent lane. As with general driving, these rewards operate on ground-truth state from offline logs.

Data generation. For each task, we run Diffusion-ES with the corresponding reward across training scenes, generating task-specific trajectory datasets. Crucially, we can generate multiple distinct behaviors for a single scene by varying the reward—increasing the incidence of rare maneuvers and synthesizing behaviors entirely absent from the original data. For non-grounded tasks, each trajectory is labeled with a fixed language description (e.g., "change lane left"). For visually-grounded tasks, we use a VLM to generate object descriptions, producing scene-specific instructions (e.g., "follow the white van").

Task-specific models. We distill the generated trajectories into two model types that do not require language. Task-specific reward modules follow the same procedure as general driving, trained per-task and selected at test time based on a discrete task specification. Fine-tuned policies are obtained by fine-tuning the flow prior on task-specific trajectories, yielding reactive policies that produce the desired behavior without reward-guided inference.

VLM-conditioned reward module. To handle language-specified instructions—including visually-grounded ones such as "follow the white van"—we train a VLM-conditioned reward model. We encode the front camera image and language instruction using a frozen Qwen2.5-VL 3B model, then cross-attend to the output token embeddings in the same manner as scene features. This enables reward prediction conditioned on visually-grounded instructions, allowing reward-guided inference to follow language commands at test time.

3.5 Experiments

We evaluate GANDALF on two axes: general driving performance and controllability. For general driving, we benchmark on NAVSIM, a planning benchmark for autonomous driving. For controllability, we evaluate task completion across two suites: maneuver-specific controllability tasks specified by discrete task labels, and visually-grounded language instruction following tasks.

We aim to answer the following questions:

- Does reward-guided inference improve over using the scene-conditioned flow prior directly as a policy?
- How do learned reward modules compare to analytic reward functions in both general driving behavior and controllability?
- Does reward-guided synthesis enable training task-specific and language-conditioned models that cannot be trained on the original data?

3.5.1 Implementation Details

We train a scene-conditioned flow prior on the NAVSIM navtrain split consisting of $\sim$103k samples. The model uses a TransFuser backbone with a ResNet-34 image encoder and a ResNet-34 LiDAR encoder, fused via a 2-layer GPT-style cross-attention module to produce BEV features at 256×256 resolution. Scene tokens are extracted from the BEV features using a 3-layer transformer decoder with 8-head attention (hidden dimension 256), which also predicts bounding boxes for up to 30 nearby agents.

For trajectory prediction, we use a flow matching decoder with a 2048-dimensional hidden space and rotary position embeddings. We predict 10-step trajectories (5-second horizon at 0.5s intervals) including heading, using 25 denoising steps with a logit-normal noise schedule. Training uses AdamW with a learning rate of 1×10^{-4}.

During reward-guided inference, Diffusion-ES samples a population of 32 trajectories, evaluates them with a reward function, selects the top 8 elites, and resamples with noise injection over 2 iterations. For the learned reward model, we train on counterfactual trajectory-reward pairs generated by replanning with analytical reward functions across the task set. The reward model shares the flow prior's scene encoder and adds a reward prediction head conditioned on trajectory tokens.

3.5.2 General Driving Performance

We evaluate on the NAVSIM benchmark using the Planning Driving Metric Score (PDMS), which captures safety, progress, and comfort. We compare three configurations of our approach against published baselines. For fair comparison, we use a standardized perception backbone across all baselines [55, 72].

Methods.

- **Flow Prior (Policy)**: The flow prior $p_\theta(\tau \mid o)$ used directly as a policy, sampling a trajectory and executing without reward-guided inference.
- **+ Analytical Reward**: Reward-guided inference using the manually designed analytical reward function.
- **+ Learned Reward**: Reward-guided inference using the learned reward model distilled via GANDALF from counterfactual generation.

Baselines. We compare against methods from the official NAVSIM benchmark, including DiffusionDrive [55].

As shown in table 3.1, the flow prior alone achieves a PDMS of 85.4, already competitive with strong baselines such as Transfuser (84.0). Introducing reward-guided inference with Diffusion-ES yields consistent improvements: the analytical reward lifts performance by 1.7 points to 88.0, surpassing Hydra-MDP (86.5). The learned reward model achieves the best result at 88.9, exceeding DiffusionDrive (88.1),

Table 3.1: Performance on NAVSIM with standardized ResNet-34 backbone (PDMS ↑).

Method	PDMS
UniAD [37]	83.4
Transfuser [72]	84.0
Hydra-MDP [51]	86.5
DiffusionDrive [55]	88.1
Flow Prior (Ours)	85.4
+ Analytical Reward	88.0
+ Learned Reward	**88.9**

the previous state of the art on this benchmark. This validates two key claims: that reward-guided inference over a flow prior outperforms using the prior directly as a policy, and that a learned reward model can outperform a hand-designed analytical reward despite requiring no privileged state at inference time. We attribute this advantage to three factors. First, analytical rewards in perception-based settings are susceptible to compounding errors from the modular perception pipeline, a challenge absent in state-based benchmarks like NuPlan where analytical rewards tend to dominate. Second, the learned reward does not need to solve the full perception problem — it only needs to capture what is relevant for distinguishing trajectory quality, a simpler target than reconstructing full scene state. Third, because the reward model is trained on trajectories generated by the flow prior itself, it is implicitly specialized to the distribution of scenarios our policy actually encounters, focusing capacity on the cases that are hardest for our planner.

3.5.3 Controllability Tasks

We evaluate five tasks: *stay in current lane*, *change lane left*, *change lane right*, *accelerate*, and *emergency stop*. We report a composite score: a binary task success indicator multiplied by PDMS, capturing both task completion and driving quality. We compare three approaches:

- **Task-Specific Reward**: Planning with a task-specific learned reward module $r_\psi^{(k)}(\tau \mid o)$.

Table 3.2: Non-grounded control task performance (Task Success × PDMS ↑).

Task	Fine-Tuned Policy	Task-Specific Reward	VLM Reward
Stay in Lane	65	**87**	86
Change Lane Left	37	**67**	63
Change Lane Right	41	**60**	**60**
Accelerate	24	**69**	65
Emergency Stop	**52**	38	40
Average	44	**64**	63

- **Fine-Tuned Policy**: A task-specific policy fine-tuned on counterfactual trajectories generated for that task.
- **VLM Reward**: Planning with the VLM-conditioned learned reward model.

table 3.2 reveals a striking advantage for planning with reward functions over policy fine-tuning. The task-specific reward achieves the highest average score (64), dominating across lane change, lane keeping, and acceleration tasks by large margins—for instance, 69 vs. 24 on acceleration and 87 vs. 65 on lane keeping. The fine-tuned policy struggles because it must learn to produce task-specific trajectories from limited counterfactual data, while the reward-guided approach leverages the full generative capacity of the flow prior and only needs to evaluate trajectory quality. Qualitative examples of reward-guided controllability are shown in fig. 3.2.

Notably, the VLM reward model—a single model conditioned on language descriptions rather than task-specific modules—achieves an average score of 63, nearly matching the task-specific reward (64). It performs comparably on lane changes and even ties on right lane change (60). The one exception is emergency stop, where the fine-tuned policy leads (52 vs. 38–40). We attribute this to the fact that emergency stopping requires generating out-of-distribution trajectories (near-zero displacement) that the flow prior rarely produces, making it difficult for any reward-guided approach to find suitable candidates in the sampled population.

3.5.4 Visually-Grounded Controllability Tasks

We evaluate two controllability tasks that require visual-language grounding: *following a vehicle* and *overtaking a vehicle*. These tasks are specified via natural language

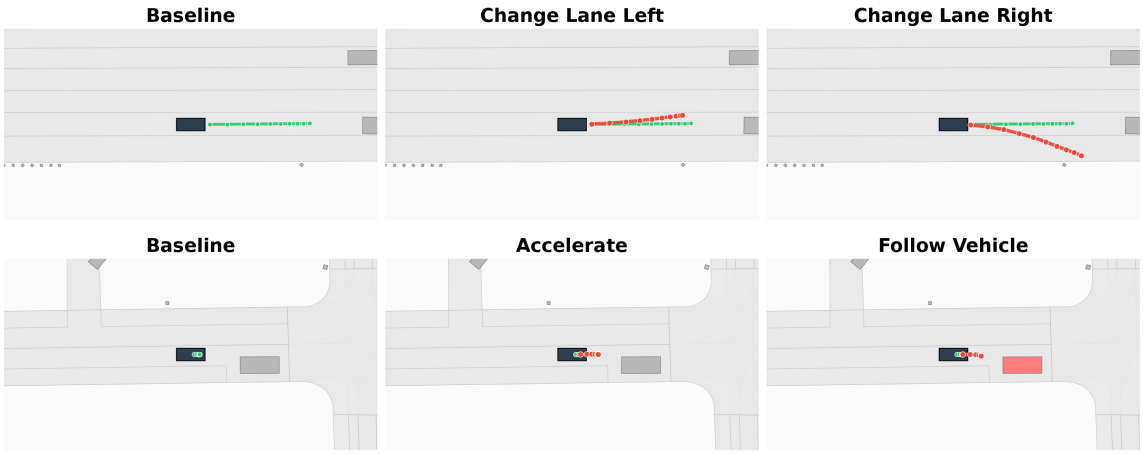


Figure 3.2: Qualitative examples of reward-guided controllability. Each row shows the same scene under different reward functions. **Top:** lane change rewards steer the ego vehicle left or right (red) relative to the baseline trajectory (green). **Bottom:** the accelerate reward increases forward progress, and the follow vehicle reward closes the gap to the lead vehicle (highlighted in red).

instructions that reference specific objects in the scene (e.g., "overtake the FedEx truck ahead"). We report the same composite score as above. We compare two approaches:

- **VLM Grounding + Analytical Reward**: A VLM parses the instruction to identify the referenced object, which is then used to parameterize an analytical reward function for planning.
- **VLM Reward**: Planning with the VLM-conditioned learned reward model, which jointly handles language understanding and reward prediction.

The VLM grounding pipeline operates as follows. Given a language instruction (e.g., "follow the white van"), we prompt a pretrained Qwen2.5-VL model with the front camera image and ask it to point at the referenced object, producing pixel coordinates via its visual grounding capability. To link this 2D point to a 3D object, we project all detected agent positions from the ego frame into image space using camera intrinsics and extrinsics, and match the nearest projected detection to Qwen's output within a pixel-distance threshold. The matched object's ego-frame position and estimated velocity then parameterize an analytical reward function—for following, a proximity reward that maintains a target distance behind the object; for overtaking,

Table 3.3: Visually-grounded control task performance (Task Success × PDMS ↑).

Task	VLM Grounding	VLM Reward
Follow Vehicle	72	**76**
Overtake Vehicle	32	**38**

a lane reward targeting the adjacent lane. Finally, Diffusion-ES optimizes against this reward. This pipeline requires three components to succeed in sequence: correct visual grounding by the VLM, accurate projection and matching to a detected agent, and a well-specified analytical reward—each introducing potential failure modes.

table 3.3 demonstrates the advantage of end-to-end learned reward prediction for visually-grounded tasks. The VLM reward model outperforms the pipeline approach (VLM grounding + analytical reward) on both tasks: 76 vs. 72 for vehicle following and 38 vs. 32 for overtaking. The pipeline approach requires a VLM to first parse the language instruction, identify the referenced object in the scene, and then map it to an analytical reward function—each step introducing potential errors. In contrast, the VLM reward model jointly reasons about language, visual grounding, and trajectory quality in a single forward pass, avoiding error propagation across modules. The gap is especially pronounced on the more challenging overtake task, where precise spatial reasoning about the target vehicle is critical.

3.5.5 Diversity–Quality Tradeoff

A key design choice in GANDALF is treating the flow model as a prior rather than a policy. When used directly as a policy, there is an inherent tension between diversity and performance: as sampling temperature increases and the model covers more of the trajectory space, output quality degrades sharply. The natural fix—reducing temperature toward determinism—recovers performance but collapses diversity, undermining the goal of generating diverse behavioral data. GANDALF breaks this tradeoff by separating the prior from reward-guided inference: the prior is free to be mode-covering and diverse, while reward guidance selects high-quality trajectories from that population.

We validate this empirically in fig. 3.3 by varying the sampling temperature from 0.05 (near-deterministic) to 2.0 and measuring PDMS across 300 scenes. At low

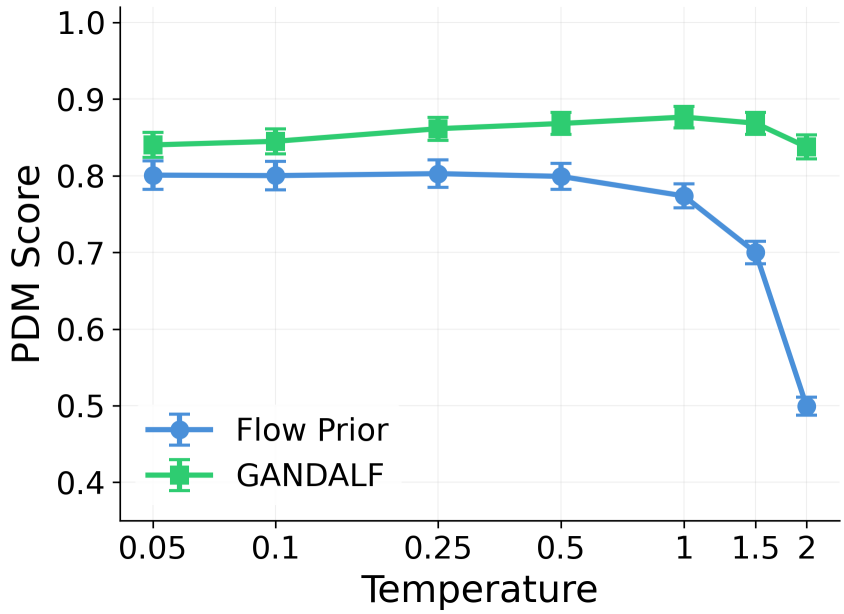


Figure 3.3: PDM score vs. sampling temperature across 300 NAVSIM scenes. At low temperatures, both approaches converge as samples become near-identical. As temperature increases, the flow prior degrades sharply while GANDALF remains stable. Error bars show standard error of the mean.

temperatures, both approaches converge—near-identical samples leave no room for reward guidance to improve upon. As temperature increases, the flow prior used directly as a policy degrades sharply, while GANDALF remains stable across the full range, retaining diverse samples without compromising on performance.

3.6 Limitations and Conclusion

Non-reactive simulation. Counterfactual trajectories are scored against recorded agent futures, assuming other agents do not react to the ego vehicle's behavior. This is reasonable for short planning horizons and mild deviations from the logged trajectory, but becomes increasingly unrealistic for aggressive maneuvers—for instance, a lane change that cuts off a vehicle which would brake in reality. Integrating reactive simulation or learned world models into the scoring pipeline could extend the framework to longer horizons and more disruptive behaviors.

Reward specification. The diversity of synthesized behaviors is bounded by the set of reward functions we define. Tasks not captured by any reward—such as navigating construction zones or responding to pedestrian gestures—cannot be generated. Scaling to open-ended behavior diversity may require learning reward functions from demonstrations or natural language specifications rather than hand-designing them, an exciting direction enabled by the VLM-conditioned reward model introduced here.

Planning cost. Reward-guided inference requires multiple forward passes through the flow prior and reward model at test time, increasing latency compared to a single-pass policy. Our temperature robustness analysis (fig. 3.3) suggests this cost is justified—direct policy sampling is brittle to hyperparameter choices that reward-guided inference absorbs gracefully—but real-time deployment at scale may benefit from distilling the planner's output distribution into an amortized policy.

Conclusion. We presented GANDALF, a framework that treats a scene-conditioned flow model as a trajectory prior and guides it with reward functions to generate diverse driving demonstrations from logged scenes. This separation of dynamics modeling from objective specification is the key enabler: the same prior supports offline counterfactual generation under varied rewards, distillation into learned reward models that match hand-designed rewards without privileged state, and training of language-controllable policies without human annotation—all from a single generative model. On NAVSIM, GANDALF achieves state-of-the-art driving performance and demonstrates visually-grounded instruction following via VLM-conditioned reward models trained entirely on synthetic data.

However, the diversity of behaviors that GANDALF can synthesize is fundamentally bounded by the set of reward functions we manually define. Scaling to open-ended behavioral diversity requires moving beyond hand-designed reward programs. In the next chapter, we address this limitation by automating the discovery of reward programs themselves, using a vision-language model to iteratively refine language programs based on visual feedback from rollout videos.

Chapter 4

EurekaDrive: Vision-Language Guided Program Search for Language-Conditioned Driving

4.1 Introduction

GANDALF showed that reward-guided planning over a generative prior can produce diverse, language-labeled driving data at scale — but it relies on manually specified reward programs, requiring human expertise to design reward functions for each desired behavior. This raises a natural question: can we automate the discovery of reward programs themselves, closing the loop on language-programmed data synthesis?

In this chapter, we address this with *EurekaDrive*, a novel framework that autonomously generates language-labeled driving data and trains a driving agent capable of performing diverse maneuvers according to user commands. Specifically, the proposed framework leverages the internet-scale pretraining and reasoning capabilities of a vision-language model (VLM) to derive reward functions for given maneuver descriptions and integrates them into a diffusion-based planner that has been pretrained on unlabeled human driving data. The VLM-guided reward enables the planner to synthesize trajectories aligned with the intended maneuvers, effectively bridging the

gap between unlabeled driving data and language-conditioned policy learning. We render environment rollouts as videos which can be directly passed to a VLM critic which is tasked with refining language instructions, allowing for the VLM to provide more detailed corrective feedback. For example, when attempting an overtaking maneuver, if the agent initially tries to change to the left lane to overtake a target vehicle but fails due to heavy traffic in that lane, the VLM critic can analyze the resulting rollout video and refine the command by suggesting the use of the right lane instead. Through such iterative feedback, the framework can generate more precise and context-aware instructions, enabling the collection of richer and higher-quality data even in complex driving scenarios.

To validate that the generated data can effectively train agents capable of performing various maneuvers, we first fine-tune a pretrained diffusion planner on the driving data corresponding to each maneuver. This process yields maneuver-specific diffusion planners specialized for tasks such as overtaking, giving way, and pulling over. Our experimental results show that fine-tuning with the generated data significantly improves the planner's success rate on the corresponding maneuver, demonstrating the effectiveness of our data generation pipeline for task-specific policy learning. Furthermore, to develop a more general language-conditioned driving agent, we train a diffusion planner over language and scene representations from a vision-language model using the language-labeled dataset created by our framework [6]. This experiment demonstrates that the proposed method enables the learning of a generalizable agent capable of generating diverse driving trajectories conditioned on natural language commands. Finally, we apply the same framework to improve general driving performance without any language supervision at test time. By using program search to generate a finetuning dataset of hard-scenario demonstrations, we show that language can serve as a powerful offline tool for data synthesis even when the deployed policy is entirely conventional, improving performance on the nuPlan benchmark.

In summary, our main contributions are as follows. (1) We propose a novel framework that leverages the reasoning capability of vision-language models to automatically derive reward functions for language-specified maneuvers and integrates them into a diffusion-based planner to generate language-labeled driving data from unlabeled human trajectories. (2) We introduce a self-improving formulation where a critic VLM refines language commands through visual feedback, enabling the

system to iteratively enhance the quality and diversity of the generated data. (3) We demonstrate that the resulting language-annotated dataset can be effectively used to train both maneuver-specific planners and a generalizable language-conditioned driving agent. Crucially, we show that using fine-tuning, we can outperform the original diffusion planner, suggesting that by iteratively repeating the process of data collection, model fine-tuning, and subsequent data generation using the improved model, the framework can progressively develop more capable and refined language-conditioned driving agents. (4) We show that program search over language-specified reward functions serves as an effective offline data synthesis tool even absent any language supervision at train or test time — generating hard-scenario demonstrations that a conventional base planner cannot produce on its own and using them to improve general driving performance on the nuPlan benchmark.

4.2 Related Work

4.2.1 End-to-end autonomous driving

End-to-end approaches for autonomous driving have been well-studied in recent years, with many methods relying on imitation learning [15, 16, 18, 37, 50, 54]. More recently, language-conditioning has been explored to train more steerable and flexible driving policies [85]; in Chapter 2, we showed that language instructions can be mapped to reward functions for test-time optimization. LMDrive [85] modifies a VLM to support language conditioning and multi-view camera inputs, showing instruction-following results in CARLA [27]. Our work builds off of Diffusion-ES (Chapter 2), using a VLM-based framework to autonomously generate corrective language instructions for guiding the lower level driving policy. There have been several efforts to train large-scale driving VLAs [4, 41, 102, 110]. However, most of these approaches assume access to language-annotated driving trajectories for training. In this work, we address how we can scalably collect such datasets via VLM-guided program search without extensive human labeling efforts.

4.2.2 Language model programs for robotics

Recent work has explored using LLMs to generate executable control code for robotic control. Code-as-policies [52] treats language models as program synthesizers that assemble robot behaviors from API calls. Other works [106] translate language instructions into reward code for MPC optimization, as does Diffusion-ES (Chapter 2), showing that rewards provide a flexible, interpretable interface for language grounding. Most similar to our work is EUREKA [60], which uses an LLM to iteratively propose and refine reward programs, discovering human-level reward designs for complex manipulation tasks. Our work differs in that we directly plan over LLM-generated rewards without requiring RL retraining at each iteration, which speeds up our search. Moreover, we rely heavily on video feedback from environment rollouts, whereas EUREKA provides feedback via text. We also demonstrate how our method can be used to generate synthetic demonstrations for fine-tuning policies downstream, allowing us to generalize to unseen scenarios.

4.2.3 Language datasets for autonomous driving

While there have been many large scale driving datasets released [11, 90], we identify the relative lack of investment into language annotations as an area of need. There have been numerous efforts to release language datasets related to driving, generally in the form of VQA datasets [4, 26, 62, 73, 87]. Our work differs from previous efforts in that our framework can generate counterfactual language-action pairs, whereas datasets which simply annotate existing driving behaviors are bottlenecked by the behavioral diversity of real driving logs. We also propose a general framework for collecting such data, which we hope will serve as a starting point for researchers to scalably annotate their own large language datasets for driving. More closely related to our work, ProSim [92] and LangTraj [14] tackle language-conditioned traffic simulation, training models to control agent behaviors in response to natural language prompts. Both rely on large-scale annotated datasets, ProSim-Instruct-520k and Inter-Drive respectively, to supervise their models. In contrast, our work targets language-conditioned *ego planning* rather than traffic simulation, and generates language annotations synthetically via VLM-guided program search, without requiring any human labeling.

4.3 Background

4.3.1 Problem setting

We are interested in policies $\pi(a_{t:t+H}|s_t, g)$ which take as input a vector-based representation of the scene s_t at timestep t as well as a language goal $g \in \mathcal{G}$, and output the future ego trajectory $a_{t:t+H}$, consisting of 2D waypoints with heading, where H denotes the planning horizon. We denote the expected return of policy π under environment dynamics p as:

$$R(\pi, g) = \mathbb{E}_{a_t \sim \pi(a_{t:t+H}|s_t,g), s_{t+1} \sim p(s_{t+1}|s_t,a_t)} \left[\sum_{t=0}^{T} r(s_t, a_t, g) \right]$$

where $r(s, a, g)$ is a goal-conditioned reward function which accounts for infractions related to safety, traffic rule compliance, and goal reaching. In this work, we consider settings where g is parameterized as a natural language instruction (e.g., "overtake the car in front", "change lanes to the right").

4.3.2 Diffusion-ES

Planning with diffusion policies. Diffusion policies model the state-conditioned distribution over ego trajectories, $\pi_\theta(a_{t:t+H} \mid s_t)$. Given an arbitrary reward function $r(s, a)$, we can use Diffusion-ES (Chapter 2) to guide the generation process via evolutionary search. Specifically, we can sample a set of N candidate actions $\{a^{(i)}_{t:t+H}\}_{i=1}^{N} \sim \pi(a_{t:t+H} \mid s_t)$, score them using $r(s, a)$, and resample around the N_{elite} highest scoring candidates using a truncated noising-denoising process, repeating this process for K iterations. For simplicity, we refer to this reward-conditioned distribution as $\pi_\theta(a_{t:t+H} \mid s_t, r)$.

Mapping language goals to reward functions. Prior work [60] has shown that code is a well-suited medium for reward functions, bridging the gap between language and robotic control. One way to map language instructions to reward code is via few-shot LLM prompting. The reward API exposes a set of primitive functions covering longitudinal and lateral control, spatial relationships between agents, and safety constraints, connecting these primitives to the control environment

and allowing reward functions to be specified as executable Python code. By providing in-context examples of language instructions paired with reward code to an LLM, we can query the LLM to generate reward code for a given language instruction, composing these primitives into arbitrarily complex maneuver-specific objectives while remaining grounded in physically meaningful quantities. For a given language goal g, we can use the above procedure to obtain a reward function r_g, which gives us a language-conditioned policy $\pi_\theta(a_{t:t+H} \mid s_t, r_g)$ which we will refer to as π_g for brevity. While this works well for relatively simple instructions or scenarios, this does not generalize well to more challenging cases which may be out-of-distribution for the provided few-shot examples. To address this limitation, we propose a reward refinement framework in which a VLM receives feedback from the simulator about previously generated reward functions and iteratively improves them. The details of this procedure are described in the following section.

4.4 Method

In this section, we first outline our proposed language-guided search framework, *EurekaDrive*. Then we describe how we use this framework to synthetically generate driving data for fine-tuning policies.

4.4.1 VLM-guided program search

We propose *EurekaDrive*, a novel framework for autonomous language-guided exploration without expert labels using VLM-guided reward search. The high-level algorithm is visualized in Figure 4.1.

Search formulation. Rather than searching over actions $a_{t:t+H}$ or learned policy parameters θ directly, we propose to search over language subgoals $g' \in \mathcal{G}$ to steer a language-conditioned diffusion policy $\pi_{g'}$. For instance, given an initial goal "overtake the car in front", an improved subgoal might be "change lanes to the left, accelerate until ahead of car 4, then change back to the original lane". There are several advantages to this approach. Searching over language instructions allows us to leverage internet-scale pretrained VLMs which already contain strong priors on reasonable reward modifications for driving. Optimizing language instructions $g' \in \mathcal{G}$

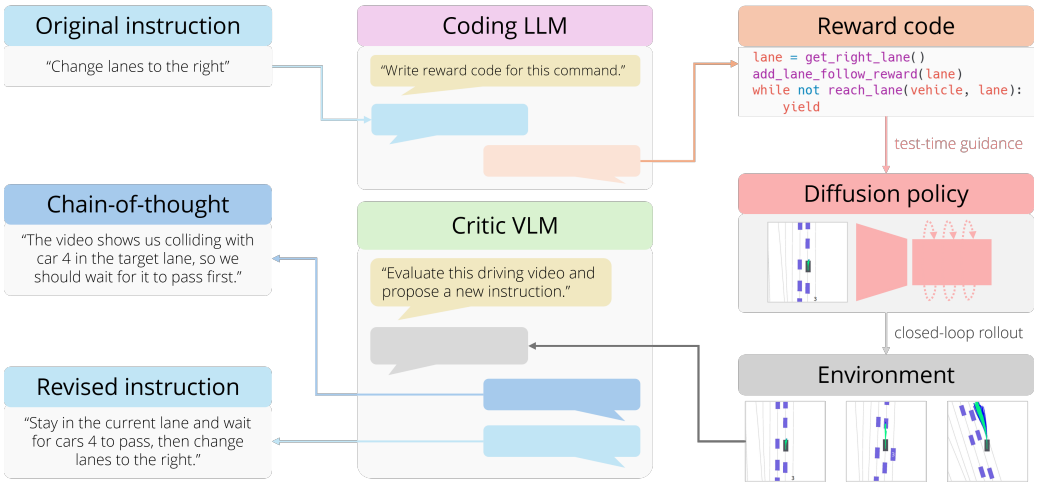


Figure 4.1: Given a driving scenario, high-level language goal, and associated reward function, EurekaDrive guides a language-conditioned diffusion planner with automated feedback from a VLM. Given an initial language instruction, an LLM is used to map this instruction to reward code, which then guides a pretrained diffusion policy in the environment. This rollout can be rendered into a video which is used to prompt a VLM for a revised instruction. EurekaDrive repeats this process until the goal is reached or a set number of iterations have run.

is also much more efficient than optimizing low-level actions directly, allowing for semantically aware exploration with fewer environment interactions.

Environment rollouts and video feedback. Given an initial goal g, we initialize a language-conditioned planner π_g as described in Section 4.3.2 and roll it out in the environment. Rather than relying solely on the scalar return $R(\pi_g, g)$ as feedback — which is sparse and uninformative when the planner fails — we propose to render the rollout as a video $V(\pi_g) = (M(s_0), \ldots, M(s_T))$, where $M(s)$ is a function that produces a rasterized BEV image from state s. To further improve the VLM's scene understanding, we augment each frame with lane contrast enhancement, per-actor numerical identifiers, and proximity-based highlighting of nearby agents, as shown in Figure 4.2.

Iterative reward refinement. Given the rollout video $V(\pi_g)$, the original instruction g, and a description of the reward API (Section 4.3.2), we prompt Gemini 2.5 Flash [21] to produce a refined instruction g' that diagnoses the failure and proposes a corrective reward function. This updated instruction is mapped to a new reward

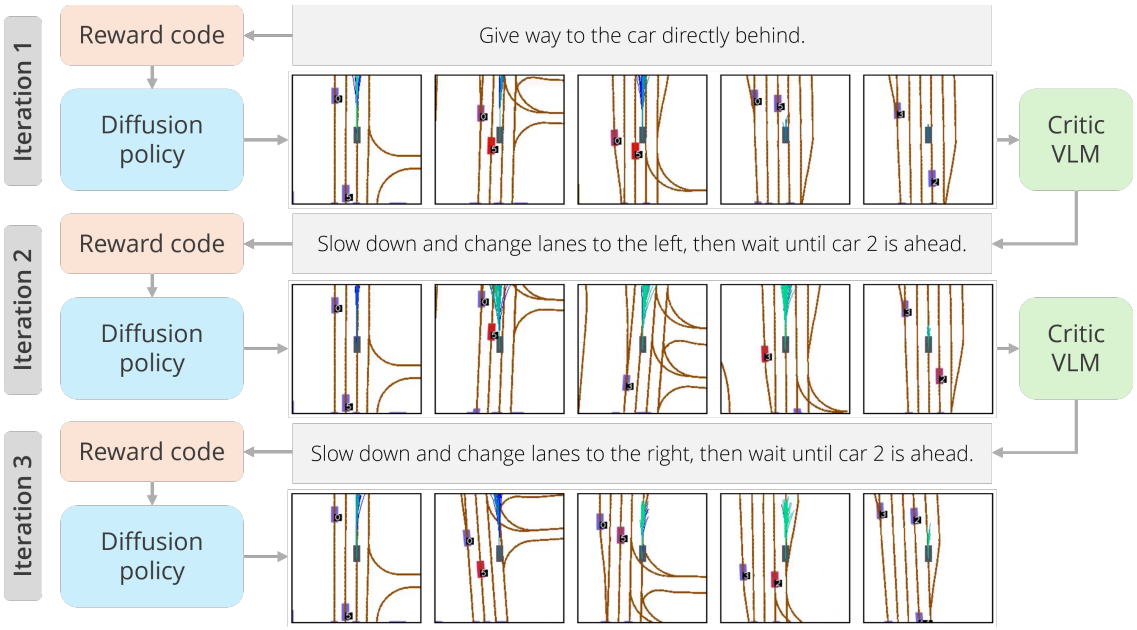


Figure 4.2: Visualization of EurekaDrive on a "give way" instruction goal. Each row represents an iteration of program search consisting of a language instruction, environment rollout, and VLM query. In this example, the initial instruction reward is too sparse and results in a collision. On the next iteration, the agent attempts to more slowly change lanes, but still collides with oncoming traffic in the left lane. Finally, the agent opts to simply change lanes to the right instead, avoiding any potential collisions.

function $r_{g'}$ via few-shot LLM prompting, and the planner $\pi_{g'}$ is re-evaluated in the environment. This process repeats until the task success criterion is met or a maximum number of iterations is reached.

4.4.2 Data collection

Language-conditioned driving. To construct the language-annotated dataset, we run *EurekaDrive* across all training scenarios for each of the six target maneuvers (follow lane, change lane left, change lane right, pull over, overtake, and give way). For each scenario, we initialize the planner with the maneuver instruction g and iterate until either the task success criterion is met or the iteration budget is exhausted. The language label assigned to each successful trajectory is the instruction g' from the iteration in which success was achieved — this may be the original g if the planner

succeeds on the first attempt, or a refined subgoal if iterative correction was required. Scenarios for which no successful rollout is found within the iteration budget are discarded.

General driving improvement. We apply the same framework to improve general driving performance without any language supervision at train or test time. We first sweep all training scenarios using vanilla Diffusion-ES under general driving rewards, and identify failures as scenarios where the task success criterion is not met. We then run *EurekaDrive* on these failure cases, replacing maneuver-specific prompts with general driving prompts that task the VLM with correcting whatever failure mode is observed in the rollout video. Successful rollouts are retained as demonstrations for finetuning a conventional driving policy. Scenarios that succeed under vanilla Diffusion-ES are excluded, as the base planner can already handle them without additional data. A detailed explanation of the prompts we used is presented in the supplementary materials.

4.4.3 Policy finetuning

Successful demonstrations collected via *EurekaDrive* are used to finetune diffusion-based driving policies. In all cases, finetuning is initialized from the same pretrained base policy π_θ (details in Sec. 4.5).

Maneuver-specific diffusion policies. For each target maneuver, we finetune a separate policy exclusively on demonstrations collected for that maneuver. This allows the diffusion model to specialize its denoising dynamics to the action distribution of individual maneuvers, which are often highly multimodal and difficult to capture with a single generalist policy.

Diffusion VLA. To develop a more generalizable language-conditioned agent, we additionally train a unified policy that conditions on both the scene state and a language instruction. Rather than specializing per maneuver, this model learns a shared representation across all maneuver classes, enabling generalization to arbitrary language-conditioned instructions at test time. Full architectural details are provided in the supplementary materials.

General driving improvement. For the general driving setting, we finetune a single unconditional diffusion policy on all hard-scenario demonstrations collected

without language supervision. Unlike the language-conditioned models above, this policy requires no language input at train or test time — the role of language is limited entirely to the offline data synthesis process.

4.5 Experiments

In this section, we present a series of experiments designed to answer the following questions:

- Can *EurekaDrive* generate driving data that aligns with given language commands?
- Can the generated language-annotated dataset be used to train language-based driving agents?
- Can our finetuned driving planners outperform the original planners used for data collection?
- Can EurekaDrive be independently applied to generate demonstrations that improve general driving performance, without relying on language supervision at test-time?

We then demonstrate two applications of EurekaDrive: (1) generating a language-annotated dataset to train language-conditioned driving agents for specific maneuvers, and (2) independently applying EurekaDrive to hard scenarios to synthesize a separate set of demonstrations for general driving improvement, without any language supervision.

4.5.1 Benchmark and Experimental Setup

Dataset and simulator In our experiments, we make use of the nuPlan dataset and simulator, which provides near-ground-truth perception of surrounding vehicles, pedestrians, lanes, and traffic lights, allowing planners to focus on decision-making rather than low-level perception. All experiments are conducted in the nuPlan closed-loop simulation setting with reactive surrounding agents. The dataset contains over $1,300$ hours of driving data with diverse driving conditions and behaviors. We used 1% of the dataset, subsampled from 20Hz to 0.5Hz, to pre-train the diffusion planner

employed in our program search framework.

Benchmark. To evaluate the performance of our model on specific maneuvers, we constructed a benchmark using test scenarios extracted from the nuPlan *reduced_val14_split*. We mainly focus on six types of maneuvers:

- **Follow lane:** The ego vehicle must follow current lane.
- **Change lane left:** The ego vehicle must change lane to left.
- **Change lane right:** The ego vehicle must change lane to right.
- **Pull over:** The ego vehicle must safely transition out of its current lane and move toward the roadside, and stop.
- **Overtake:** The ego vehicle must change lanes and accelerate to pass a target vehicle positioned ahead.
- **Give way:** The ego vehicle must yield to a target vehicle approaching from behind by changing lanes and decelerating.

For each maneuver type, we selected 50 test scenarios in which the maneuver can be feasibly executed. To ensure the feasibility of each scenario, we performed 10 iterations of program search on every scenario in the *reduced_val14_split* and collected only those in which the specified maneuver was successfully completed at least once. Note that the program search was conducted solely for feasibility checking, and the trajectories generated during this process were not used for model training. This procedure guarantees that the resulting benchmark consists of realistic and executable driving scenarios, enabling reliable evaluation of maneuver-specific success rates under consistent simulation conditions. This also highlights that *EurekaDrive* can be used more generally as a systematic tool for constructing task-specific, feasibility-verified benchmarks for evaluating driving agents.

4.5.2 Program Search for Maneuver Generation

To show that *EurekaDrive* can generate appropriate reward functions that enable the agent to perform each command according to the given situation, we demonstrated *EurekaDrive* for three complex maneuvers (pull over, overtake, and give way). To enable this process, we provided the VLM with a set of primitive reward APIs, such as following a lane, increasing or decreasing speed, and computing relative position to

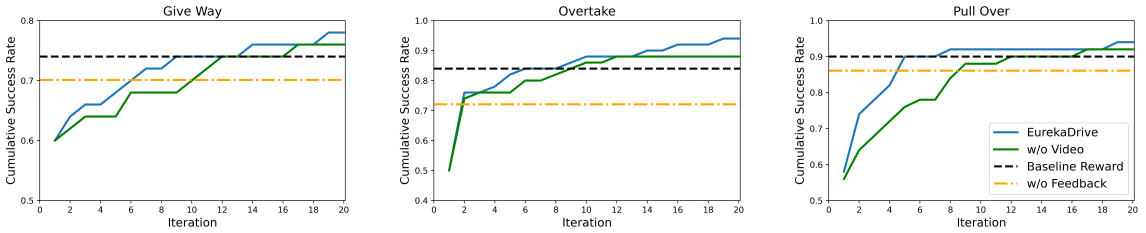


Figure 4.3: Cumulative success rate per iteration on three complex maneuvers. Compared to using the **baseline reward** or **VLM without any feedback**, *EurekaDrive* was able to discover a larger number of successful demonstrations through sequentially updating the reward function. Although the **VLM without driving videos** eventually approached a success rate similar to *EurekaDrive*, providing video feedback enabled the framework to identify suitable reward functions more rapidly, demonstrating the importance of visual context in accelerating the search process.

a target vehicle. Figure 4.2 illustrates how *EurekaDrive* progressively refines both the language command and the corresponding reward function through iterative feedback from driving videos. As the iterations proceed, the framework becomes increasingly capable of generating precise and context-aware reward functions, ultimately enabling the agent to successfully execute the given maneuver.

To further evaluate the efficiency of *EurekaDrive* in discovering effective reward functions, we conducted an ablation study comparing it with several baselines:

- **Baseline Reward**: A hand-designed reward function that is specific to each maneuver but fixed across all driving conditions, without adapting to environmental context. This comparison highlights that the VLM can generate more precise and context-aware reward functions than manually engineered ones by incorporating information about the current driving environment.
- **w/o Video**: The VLM does not receive driving videos and thus lacks access to environmental conditions. It only obtains binary feedback on whether the rollout under the generated reward function succeeded in the given iteration. This comparison demonstrates that the VLM does not simply generate reward functions at random, but rather leverages visual information from the driving video to explore the reward space more efficiently.
- **w/o Feedback**: The VLM receives no feedback from the simulator and instead generates generic reward functions for each task, independent of execution

outcomes. This comparison shows that incorporating feedback from previous rollouts encourages the VLM to explore a broader and more informative range of reward functions, ultimately improving its ability to discover the reward functions.

For both the **Baseline Reward** and **w/o Feedback** settings, we performed 20 independent runs under identical conditions and compared the cumulative success rates across all maneuvers. As shown in Figure 4.3, *EurekaDrive* discovers successful driving trajectories more rapidly and consistently than the baselines, demonstrating its superior efficiency in learning task-aligned reward functions through iterative feedback. A detailed explanation of designing baseline rewards and primitive reward APIs is presented in the supplementary materials.

4.5.3 Training Language-Conditioned Driving Agents

To construct our annotated dataset, we first sampled 6,641 training scenarios from the nuPlan dataset. For complex maneuvers, such as pull over, overtake, and give way, we applied three iterations of program search and retained only the successful rollouts as demonstrations. For comparatively simple maneuvers, such as follow lane and change lane, demonstrations were generated using basic manually designed baseline rewards. While these maneuvers typically do not require detailed reward shaping to produce valid behaviors, and program search was not necessary, including these samples helps provide coverage of common driving behaviors and supports the training of a more generalizable driving agent. Table 4.1 summarizes, for each maneuver, the proportion of successful demonstrations obtained relative to the number of scenarios used for data synthesis, illustrating the effectiveness of *EurekaDrive* across different behavior types.

Table 4.1: Success rate during annotated data collection.

Maneuver	**Success Rate**
Follow Lane	0.69
Change Lane Left	0.48
Change Lane Right	0.51
Pull Over	0.61
Overtake	0.11
Give Way	0.39

Using the annotated dataset constructed through our program search framework, we trained both the Finetuned Diffusion-ES models and the Diffusion VLA model. For the Finetuned Diffusion-ES models, we further fine-tuned the diffusion planner used during program search on the dataset corresponding to each maneuver. For the Diffusion VLA model, we trained a single model across all maneuvers by conditioning the diffusion planner on BEV images and language commands encoded using the Qwen3-VL model. The trained models were evaluated against the following baselines:

- **Diffusion-ES**: The diffusion planner is not finetuned for any maneuver and instead relies on test-time guidance with manually designed baseline reward functions to complete each task.
- **PDM-closed**: a conventional planner that uses heuristics to generate trajectory proposals and scores them using a manually crafted reward function. Our implementation is based on the PDM-closed planner proposed in [23], with additional lateral offsets applied to the trajectory proposals to enable lane-change maneuvers.

We also compared the performance of each diffusion-based planner with and without evolutionary search (ES), using $K = 5$ and $K = 0$ of iterations, respectively. In each iteration of ES, the planner generate 64 candidate trajectories, selected the top 8 elite trajectories, and used them to construct the proposal for the next iteration. We would like to note that the original diffusion planner, being unable to generate task-specific trajectories on its own, also produced 64 candidate trajectories even under the $K = 0$ setting and executed the one with the highest reward sum. For

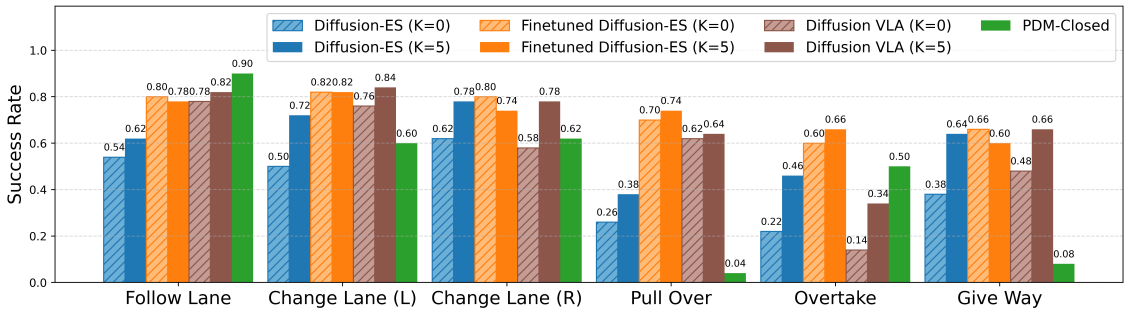


Figure 4.4: Comparison of maneuver success rates across different planners and training settings. The results illustrate the performance of Diffusion-ES, Finetuned Diffusion-ES, Diffusion VLA, and PDM-closed under varying ES iteration budgets across six maneuver types. Finetuned Diffusion-ES achieves consistently higher success rates than the untuned baseline, while Diffusion VLA also attains comparable performance.

fairness, the same procedure was applied to both the Finetuned Diffusion-ES and the Diffusion VLA when evaluating their $K = 0$ performance.

Our comparative results are presented in Figure 4.4, and the key findings are summarized as follows.

PDM-closed vs. Finetuned Diffusion-ES: PDM-closed achieves strong performance for the follow lane maneuver. However, for the other maneuvers, its performance is consistently lower than that of the fine-tuned models. In particular, when lane changes are required in traffic-dense environments, PDM-closed often fails to generate sufficiently detailed trajectory proposals that properly account for surrounding vehicles. As a result, the planner struggles to produce feasible and context-aware maneuvers. In contrast, the fine-tuned models generate trajectory proposals that better reflect both the surrounding traffic conditions and the specified maneuver, resulting in improved performance on these more complex driving scenarios. Furthermore, the finetuned Diffusion-ES models consistently improve upon these results, demonstrating that diffusion-based planners can be adapted to complex, maneuver-specific behaviors when provided with targeted demonstrations generated by *EurekaDrive*.

Diffusion-ES vs. Finetuned Diffusion-ES: The diffusion planners finetuned with maneuver-specific demonstrations consistently achieve higher success rates than their

non-finetuned counterparts across all maneuvers. Notably, the finetuned Diffusion-ES maintains strong performance even with a small number of ES iterations ($K = 0$), exhibiting only a minor performance gap relative to configurations with more iterations. In contrast, the original Diffusion-ES shows a much larger performance variance depending on the number of ES iterations, with substantially lower success rates when the ES budget is limited. These results indicate that the finetuned planners generate candidate trajectories that are inherently better aligned with the targeted maneuver, reducing reliance on extensive ES and demonstrating the practical value of training with program-search-generated demonstrations.

Diffusion-ES vs. Diffusion VLA: The Diffusion VLA trained on the program-search-generated dataset achieves success rates that are comparable to those of the maneuver-specific Finetuned Diffusion-ES across most tasks. Remarkably, for several maneuvers, the Diffusion VLA attains higher success rates with a minimal evolutionary search budget ($K = 0$) than the Diffusion-ES baseline even when the latter is allowed a larger number of ES iterations ($K = 5$). This demonstrates that the dataset produced by *EurekaDrive* is not only effective for maneuver-specific fine-tuning but also sufficiently rich and diverse to train a generalizable, language-conditioned driving agent. Overall, the results confirm that Diffusion VLA can leverage the synthesized language-annotated trajectories to perform a wide range of maneuvers with high reliability and reduced dependence on computationally expensive evolutionary search.

Latency comparison: Table 4.2 reports the latency of the two model classes under different planning budgets ($K = 0$ and $K = 5$). Notably, the finetuned conditional diffusion model achieves substantially lower latency in the zero-ES setting ($K = 0$) compared to Diffusion-ES with iterative refinement at $K = 5$ (1.1 s vs. 4.2 s), while still outperforming it in success rate on complex maneuvers (Fig. 4.4).

4.5.4 Program Search for General Driving Improvement

Data Generation We apply *EurekaDrive* to the same training scenarios used in the language experiments, but without any maneuver-specific supervision. First, we sweep through all training scenarios using vanilla Diffusion-ES with general driving rewards, and identify failures as scenarios where the nuPlan score falls below a fixed threshold. We then run *EurekaDrive* for four iterations on these failure cases, using

Table 4.2: Planning latency per step. Diffusion-ES latency is identical for both finetuned and non-finetuned variants.

Method	**Latency (s)**
Diffusion-ES (K=0)	1.1 ± 0.06
Diffusion-ES (K=5)	4.2 ± 0.59
Diffusion VLA (K=0)	4.8 ± 0.19
Diffusion VLA (K=5)	6.3 ± 0.34

the same VLM-guided reward search loop described in Section 4.4. Crucially, rather than maneuver-specific prompts and rewards, the VLM is provided only with general driving reward primitives and tasked with writing a reward function that corrects the observed failure, using visual feedback from the BEV rollout video. Successful rollouts produced during this process are retained as demonstrations, yielding a dataset of hard-scenario trajectories on which we finetune the diffusion planner.

Results We evaluate the finetuned planner on the nuPlan *reduced_val14_split* and compare against PDM-Closed and the non-finetuned Diffusion-ES baseline. As shown in Table 4.3, finetuning on *EurekaDrive*-generated demonstrations yields a driving score of 0.93, outperforming both the non-finetuned Diffusion-ES (0.91) and PDM-Closed (0.92). These results demonstrate that *EurekaDrive* can serve as a general-purpose data synthesis tool beyond language-conditioned driving, producing demonstrations on hard scenarios that meaningfully improve overall driving performance.

Table 4.3: Driving score comparison on nuPlan benchmark using *reduced_val14_split*.

Method	**Driving Score**
PDM-Closed	0.92
Diffusion-ES	0.91
Diffusion-ES (Finetuned)	0.93

4.6 Conclusion

We introduced *EurekaDrive*, a scalable framework for generating language annotated driving data using VLM-guided program search over reward functions. Through iterative refinement using visual feedback from rollout videos, the framework synthesizes high-quality demonstrations for complex maneuvers without manual annotation. We show that finetuning on this data significantly improves maneuver success rates, and that a unified Diffusion VLA trained on the resulting dataset generalizes across diverse language-conditioned instructions. Beyond language-conditioned driving, we demonstrate that the same framework improves general driving performance by synthesizing hard-scenario demonstrations that the base planner cannot produce on its own, highlighting the broader utility of language as an offline tool for data synthesis in autonomous driving.

Chapter 5

Latent Preference Finetuning for Driving Policies

5.1 Introduction

In the previous work, we showed how online natural language instructions can be used to guide driving agents. However, we are interested in how human feedback can be used to improve policies over time. In this work, we propose initial work towards improving driving policies from offline human preference feedback.

Rather than learn directly from open-ended natural language feedback, we focus on learning from preferences over discrete high-level intentions (e.g., "change lanes to the left", "slow down", "accelerate") for a few reasons. First, it is significantly faster and easier for human labelers to select the most desirable high-level intent versus asking for a natural language correction. In special cases, it may even be possible to extract such labels programmatically without relying on manual human labeling. Next, since the intent-space is discrete and enumerable, learning models over these preferences is much easier than modeling preferences over unstructured text. We can also assume that the human can identify the global "best" intent, as opposed to in conventional RLHF settings where preference labels are only locally relevant and require more complex methods for extracting usable reward models.

We propose a novel technique for fine-tuning driving policies using human latent

preferences (i.e., preferences over high-level intents). Given a two-stage policy which consists of a high-level latent policy and a low-level policy conditioned on a high-level latent, we use preference labels to supervise the high-level policy using intent-specific high-level policies. This allows us to refine driving policies using only high-level feedback. We find that these procedure leads to improved driving performance when tested on real-world driving tasks. As this work was conducted in an industry setting, quantitative results are omitted due to proprietary restrictions.

5.2 Related Work

There has been substantial work in aligning language models with human preferences [68, 74]. In the conventional RLHF setting, preference data is generated by sampling completions from a pretrained model and asking a human labeler to rank them. This is usually done after pretraining on large amounts of internet-scale data and supervised fine-tuning on task-specific data. Since preference labels provide less training signal than imitating "ground-truth" samples, these methods rely on theoretical preference models such as Bradley-Terry [8] to extract reward models [68] or policy gradients [74]. As mentioned earlier, we choose to focus on categorical intent preferences in this work due to ease of optimization and data collection.

Separately, there is recent work in building interactive robotic agents which can improve from human feedback [56, 86, 96]. Most closely related to our approach is YAY Robot [86], which uses human language supervision to guide a low-level language-conditioned policy for manipulation. While the motivation is similar, we opt for categorical intent latents due to the difficulty of collecting large amounts of high-quality language corrections in our setting. Training language-conditioned driving policies is highly non-trivial and we leave this to future work. Another related work is SkiP [96], which uses human feedback both for skill extraction and policy improvement. However, SkiP focuses on a preference learning setup which is more reminiscent of prior work in RLHF as the human not asked to uniquely identify the best skill to execute. One major difference which differentiates our work from the above is also that whereas they mostly consider smaller datasets on the order of thousands of samples, we scale up our approach to handle several orders of magnitude more data.

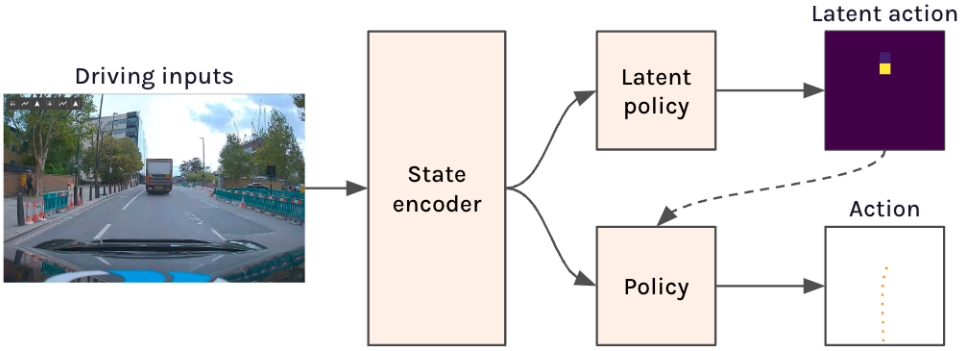


Figure 5.1: Architecture diagram for end-to-end two-stage driving policy.

5.3 Method

5.3.1 Model Architecture

The architecture for our driving policy is outlined in Figure 5.1. Our observation o consists of driving inputs (e.g., camera images, route map), with two past frames included for temporal context. Additionally, we include frames from three front-facing cameras to cover a wider field of view. Our policy processes the observation with a learned state encoder backbone and uses these state features for both the high-level latent policy and the low-level policy. This two-stage policy uses discretized 2-second waypoint goals z_{wp} as intermediate latent actions (i.e., the (x, y) position of the ego-vehicle after 2 seconds). We assume an 11×11 binning of waypoint goals in log-polar coordinates. Our low-level actions a are 2D waypoint trajectories sampled at 5 Hz with a horizon of 2 seconds, so we can extract our high-level latent z_{wp} by taking the last waypoint in the low-level action and applying binning. The high-level policy $\pi_{wp}(z_{wp}|o)$ is trained to predict the ground-truth waypoint goal bin, which amounts to cross-entropy minimization. The low-level policy $\pi_{out}(a|o, z_{wp})$ takes the ground-truth latent action as input is trained to predict a Gaussian which maximizes the likelihood of the ground-truth trajectory action.

We train the above model using behavior cloning and measure the performance of our proposed finetuning method with respect to this base policy. During preference finetuning, we freeze the state encoder and low-level policy and only finetune the high-level policy.

5.3.2 Intent Policy Finetuning

In this section we discuss how we train intent-specific latent policies, as well as how those policies are used for latent preference finetuning. We introduce an intent latent:

$$z_{int} \sim \text{Categorical}(K)$$

where there are K total possible mutually exclusive intents, each representing a high-level behavior such (e.g., "overtaking", "slowing", "accelerating"). While we could theoretically perform preference finetuning using the spatial waypoint latents introduced previously, we found that it was challenging for human annotators to provide high-quality labels for spatial waypoints. On the other hand, high-level intents are easier to interpret and label due to their semantic nature.

Using the above, we construct an intent-conditioned high-level policy. In practice, this can be implemented by training a mixture of K high-level policies, one for each intent:

$$\pi(z_{wp}|o, z_{int}) = \sum_i \mathbb{1}(z_{int} = i)\pi(z_{wp}|o, z_{int} = i)$$

We find that using a single shared policy can lead to the intent-conditioning not being used, whereas training separate policies avoids collapse.

This suggests a natural scheme for finetuning using preference labels over intents. For a given intent label $z_{int} = i$, we can use our intent-conditioned high-level policy to generate a new target $\hat{z}_{wp} \sim \pi(z_{wp}|o, z_{int} = i)$ for supervising the high-level policy in our original model. Note that this allows us to generate labels for trajectories where the demonstrations are not expert as well.

5.4 Results

Experiments were conducted on a large-scale internal driving dataset. However, as this work was performed in an industry setting, quantitative results and detailed experimental findings are omitted due to proprietary restrictions.

Chapter 6

Conclusion

A long-standing challenge in autonomous driving is that the space of possible situations grows faster than any dataset can cover. While learning-based planners have made remarkable progress on common driving scenarios, their performance degrades precisely where it matters most — on the rare, safety-critical edge cases that naturalistic data struggles to represent. This thesis has argued that language supervision, and in particular language programs, offer a practical path forward: rather than waiting for rare behaviors to appear in data, we can specify them programmatically, generate demonstrations from them, and distill those demonstrations into policies that improve over time — all without expensive human annotation.

6.1 Summary of Contributions

We began with Diffusion-ES (Chapter 2), which established the core mechanism underlying all subsequent work: combining a pretrained diffusion trajectory prior with evolutionary search over language-programmed reward functions. By decoupling the generative prior from the reward objective, Diffusion-ES can optimize arbitrary black-box objectives at test time — including those synthesized from natural language instructions via few-shot LLM prompting — without any retraining. This enabled zero-shot instruction following on challenging nuPlan scenarios while matching the performance of PDM-Closed, a carefully engineered sampling-based planner. The fundamental limitation, however, was that the policy itself learned nothing perma-

nently: every scenario required expensive iterative search at test time, and oracle language instructions were assumed available at deployment.

GANDALF (Chapter 3) addressed this by shifting reward-guided inference from online planning to offline data synthesis. Rather than solving scenarios one-by-one at test time, GANDALF runs reward-guided generation across large numbers of logged scenes, producing diverse counterfactual trajectories that are then distilled into learned reward models and policies. A key finding was that treating the flow model as a prior rather than a policy breaks the fundamental tension between diversity and quality: the prior is free to be mode-covering, while reward guidance selects high-quality candidates from that diverse population. The distilled learned reward model achieved state-of-the-art performance on NAVSIM, and the same framework extended naturally to language controllability by labeling generated trajectories programmatically from reward semantics. The remaining bottleneck was that behavioral diversity was bounded by the set of manually specified reward functions.

EurekaDrive (Chapter 4) closed this loop by automating the discovery of reward programs themselves. A vision-language model watches rollout videos and iteratively refines language programs, discovering solutions to hard edge cases that the base planner cannot handle on its own. Finetuning on EurekaDrive-generated demonstrations yielded a policy that outperformed both PDM-Closed and the non-finetuned Diffusion-ES baseline on the nuPlan benchmark, and the resulting Diffusion VLA trained on language-annotated data generalized across diverse maneuver types — suggesting that the synthesized data is rich enough to support a generalizable language-conditioned agent. Together, these three works trace a progression: from test-time guidance, to offline distillation, to automated program discovery — each step reducing the reliance on human specification while expanding the behavioral coverage of the resulting policies.

Chapter 5 explored a complementary direction: rather than specifying desired behaviors programmatically, latent preference finetuning proposes to refine them from human preferences over high-level driving intents. Where language programs enable top-down specification of behaviors we lack demonstrations for, preference supervision enables bottom-up refinement based on human judgments about behaviors we have already observed. Together these directions suggest a broader design space for semantic supervision of driving policies.

6.2 Limitations

Several limitations of the current work are worth being direct about.

The most significant is simulation dependence. GANDALF scores counterfactual trajectories against recorded agent futures, assuming other agents do not react to the ego vehicle's behavior — a reasonable approximation for mild maneuvers but increasingly inaccurate for aggressive ones. EurekaDrive, on the other hand, relies on accurate reactive simulation to provide meaningful visual feedback to the VLM critic. This work does not address the sim-to-real gap: how well policies trained and evaluated in these simulation environments transfer to real deployment conditions, with all the attendant noise in perception, actuation, and agent behavior, remains an open question.

A second limitation is that language programs are only as expressive as the API they operate over. Gaps in the reward interface lead to program failures that the VLM cannot easily diagnose or correct, and maintaining and extending the interface requires ongoing manual effort — not unlike tuning the reward functions of a conventional planner. More fundamentally, some behaviors are inherently easier to learn bottom-up from demonstrations than to specify top-down. Highly dynamic and socially interactive maneuvers — tight merges in dense traffic, implicit negotiation at uncontrolled intersections — can be difficult to capture cleanly in code, and for these cases a purely programmatic approach may not be the most natural fit.

Finally, the program search work largely operates over structured perception outputs: bounding boxes, lane graphs, agent velocities. This representation misses perceptual cues that matter in real driving but are difficult to encode in a scene graph — a driver's facial expression signaling intent to yield, a pedestrian's body language at a crosswalk, informal social negotiation between road users. Extending the approach to richer perceptual grounding is a prerequisite for handling the full range of interactions that a deployed system would encounter.

6.3 Future Directions

Several extensions follow naturally from the current work.

The most direct is extending EurekaDrive to raw sensor inputs. Currently, program

search operates over structured perception outputs, which limits both the behaviors it can reason about and the scenarios it can handle. Building a programmatic interface suited to visual queries — for instance, checking whether a vehicle's turn signal is active, or estimating a pedestrian's heading from image features — would allow reward programs to reason over richer perceptual evidence. Code-based approaches to visual reasoning such as ViperGPT have demonstrated that language models can compose perception primitives into complex visual queries, and a similar interface for driving would also directly address the scene representation limitation described above.

A second direction is extending program search beyond ego control. EurekaDrive currently generates language-conditioned behaviors for the ego vehicle alone; other agents follow their recorded or simulated trajectories. Controlling additional agents with language-specified goals would enable synthesis of tightly coupled interactive scenarios — a merging vehicle that adjusts its speed in response to the ego, an adversarial agent that probes the ego's responses — which are precisely the kind of rare, interactive situations that naturalistic data lacks and that are hardest for current policies to handle.

The language annotation infrastructure developed in this work also has unrealized potential beyond task-level conditioning. The reward programs used to generate each trajectory contain rich structure: preconditions, intermediate goals, sequential logic, and explicit reasoning about other agents. This structure could be used to automatically annotate trajectories with chain-of-thought reasoning traces, providing a form of supervision that encodes not just what the vehicle does but why — an approach that has shown promise in robotics and may improve generalization and interpretability in driving as well.

It is also worth revisiting the distillation step. Behavioral cloning on EurekaDrive-generated demonstrations is a natural starting point, but it does not make full use of the reward labels that are already available for every generated trajectory. Reinforcement learning-based finetuning is one alternative, and preliminary experiments in this direction suggest it has promise, though getting it to work reliably remains a challenge. More broadly, finding distillation methods that better exploit the contrastive structure of the generated data — trajectories spanning a wide range of quality under the same reward function — seems like a fruitful direction.

Finally, all of the experiments in this thesis were conducted in the nuPlan and CARLA simulation environments, which are behaviorally limited relative to real-world driving. Investigating these ideas at scale — in larger, more naturalistic datasets with greater diversity of scenarios, geographies, and driving cultures — is an important step toward understanding how well the approach generalizes and where its limits lie.

6.4 Broader Impact and Closing Remarks

The framing of the long-tail problem in autonomous driving has often been one of data quantity: if we could only collect enough miles, enough edge cases, enough rare events, the problem would eventually be solved. The work in this thesis suggests a different framing. The long tail is not primarily a quantity problem — it is a specification problem. The scenarios that matter most are rare precisely because they are unusual, and collecting enough naturalistic demonstrations to cover them may simply not be possible. What is needed is a way to specify the behaviors we want, generate them on demand, and verify that the resulting policies behave correctly — without waiting for the world to hand us the right data.

Language programs offer one interface for this kind of specification. They are precise enough to generate consistent behavior, flexible enough to express a wide range of objectives, and grounded in the same semantic structure that humans use to reason about driving. The results here suggest that this interface, combined with generative trajectory models and automated feedback from vision-language models, is already practical: it achieves competitive performance on established benchmarks, requires no human annotation of driving behaviors, and produces policies that improve beyond their training data.

There is something broader here too. The question of how to supervise learned behavior at scale — how to communicate what we want a system to do in situations it has never encountered — is not unique to autonomous driving. It arises whenever a learned policy must operate in an open world with a long tail of rare situations. Language, in its various forms, seems like a natural part of the answer: not as a replacement for demonstration data, but as a complement that encodes intent, enables counterfactual specification, and connects human knowledge to learned behavior in a way that raw trajectories cannot. This thesis has tried to make that case concretely,

in a domain where the stakes are high and the long tail is vast. We hope it contributes, in a small way, to the broader project of making autonomous systems that are not just capable in common cases, but reliably so when it counts.

Bibliography

[1] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B. Tenenbaum, T. Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision-making? *ArXiv*, abs/2211.15657, 2022. 2.1

[2] Anurag Ajay, Yilun Du, Abhi Gupta, Joshua B Tenenbaum, Tommi S Jaakkola, and Pulkit Agrawal. Is conditional generative modeling all you need for decision making? In *International Conference on Learning Representations (ICLR)*, 2023. 3.2

[3] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 3.2

[4] Hidehisa Arai, Keita Miwa, Kento Sasaki, Kohei Watanabe, Yu Yamaguchi, Shunsuke Aoki, and Issei Yamamoto. Covla: Comprehensive vision-language-action dataset for autonomous driving. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1933–1943. IEEE, 2025. 4.2.1, 4.2.3

[5] Mayank Bansal, Alex Krizhevsky, and Abhijit S. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. *CoRR*, abs/1812.03079, 2018. URL http://arxiv.org/abs/1812.03079. 2.2

[6] Kevin Black, Noah Brown, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Lucy Xiaoyang Shi, James Tanner, Quan Vuong, Anna Walling, Haohuan Wang, and Ury Zhilinsky. π_0: A vision-language-action flow model for general robot control, 2024. URL https://arxiv.org/abs/2410.24164. 4.1

[7] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathieu Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint*

arXiv:1604.07316, 2016. 2.2, 3.2

[8] Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. 5.2

[9] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:2307.15818*, 2023. 3.2

[10] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020. 2.2

[11] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020. 1, 4.2.3

[12] Holger Caesar, Juraj Kabzan, Kok Seang Tan, Whye Kit Fong, Eric Wolff, Alex Lang, Luke Fletcher, Oscar Beijbom, and Sammy Omari. Nuplan: A closed-loop ml-based planning benchmark for autonomous vehicles, 2022. 1, 2.1, 2.4, 3.2

[13] João Carvalho, An T. Le, Mark Baierl, Dorothea Koert, and Jan Peters. Motion planning diffusion: Learning and planning of robot motions with diffusion models. *ArXiv*, abs/2308.01557, 2023. 2.2

[14] Wei-Jer Chang, Wei Zhan, Masayoshi Tomizuka, Manmohan Chandraker, and Francesco Pittaluga. Langtraj: Diffusion model and dataset for language-conditioned trajectory simulation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 26622–26631, 2025. 4.2.3

[15] Dian Chen and Philipp Krähenbühl. Learning from all vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17222–17231, 2022. 2.2, 4.2.1

[16] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on robot learning*, pages 66–75. PMLR, 2020. 2.2, 4.2.1

[17] Cheng Chi, Siyuan Feng, Yilun Du, Zhenjia Xu, Eric Cousineau, Benjamin

Burchfiel, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *RSS*, 2023. 2.1, 2.2, 2.4.1, 3.2

[18] Kashyap Chitta, Aditya Prakash, Bernhard Jaeger, Zehao Yu, Katrin Renz, and Andreas Geiger. Transfuser: Imitation with transformer-based sensor fusion for autonomous driving. *IEEE transactions on pattern analysis and machine intelligence*, 45(11):12878–12895, 2022. 4.2.1

[19] Paul F Christiano, Jan Leike, Tom Brown, Miljan Marber, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017. 3.2

[20] Felipe Codevilla, Matthias Müller, Antonio López, Vladlen Koltun, and Alexey Dosovitskiy. End-to-end driving via conditional imitation learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2018. 2.2, 3.2

[21] Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025. 4.4.1

[22] Alexander Cui, Abbas Sadat, Sergio Casas, Renjie Liao, and Raquel Urtasun. Lookout: Diverse multi-future prediction and planning for self-driving. *CoRR*, abs/2101.06547, 2021. URL https://arxiv.org/abs/2101.06547. 2.2

[23] Daniel Dauner, Marcel Hallgarten, Andreas Geiger, and Kashyap Chitta. Parting with misconceptions about learning-based vehicle motion planning, 2023. (document), 2.1, 2.2, 2.4.1, 2.4.1, ??, 2.1, 2.4.1, 2.4.1, 2.4.2, 3.2, 4.5.3

[24] Daniel Dauner, Marcel Hallgarten, Tianyu Li, Xinshuo Weng, Zhiyu Huang, Zetong Yang, Hongyang Li, Vladlen Koltun, and Andreas Geiger. NAVSIM: Data-driven non-reactive autonomous vehicle simulation and benchmarking. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. 3.4.1

[25] Prafulla Dhariwal and Alex Nichol. Diffusion models beat gans on image synthesis. *ArXiv*, abs/2105.05233, 2021. 2.2

[26] Kairui Ding, Boyuan Chen, Yuchen Su, Huan-ang Gao, Bu Jin, Chonghao Sima, Wuqiang Zhang, Xiaohui Li, Paul Barsch, Hongyang Li, et al. Hint-ad: Holistically aligned interpretability in end-to-end autonomous driving. *arXiv preprint arXiv:2409.06702*, 2024. 4.2.3

[27] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017. 3.2, 4.2.1

[28] Yilun Du, Conor Durkan, Robin Strudel, Joshua B. Tenenbaum, Sander Diele-

man, Rob Fergus, Jascha Sohl-Dickstein, Arnaud Doucet, and Will Grathwohl. Reduce, reuse, recycle: Compositional generation with energy-based diffusion models and mcmc, 2023. 2.1

[29] Yilun Du, Mengjiao Yang, Bo Dai, Hanjun Dai, Ofir Nachum, Joshua B. Tenenbaum, Dale Schuurmans, and P. Abbeel. Learning universal policies via text-guided video generation. *ArXiv*, abs/2302.00111, 2023. 2.2

[30] Haoyang Fan, Fan Zhu, Changchun Liu, Liangliang Zhang, Li Zhuang, Dong Li, Weicheng Zhu, Jiangtao Hu, Hongye Li, and Qi Kong. Baidu apollo em motion planner. *arXiv preprint arXiv:1807.08048*, 2018. 2.2

[31] Katerina Fragkiadaki, Jonathan Huang, Alex Alemi, Sudheendra Vijayanarasimhan, Susanna Ricco, and Rahul Sukthankar. Motion prediction under multimodality with conditional stochastic networks, 2017. 2.2

[32] Nikolaos Gkanatsios, Ayush Jain, Zhou Xian, Yunchu Zhang, Christopher Atkeson, and Katerina Fragkiadaki. Energy-based Models are Zero-Shot Planners for Compositional Scene Rearrangement. In *Robotics: Science and Systems*, 2023. 2.1

[33] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14953–14962, June 2023. 2.2

[34] Dirk Helbing and Benno Tilch. Generalized force model of traffic dynamics. *Physical review E*, 58(1):133, 1998. 2.4.1

[35] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020. 2.2, 2.3.1

[36] Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv preprint arXiv:2309.17080*, 2023. 2.2, 3.2

[37] Yihan Hu, Jiazhi Yang, Li Chen, Keyu Li, Chonghao Sima, Xizhou Zhu, Siqi Chai, Senyao Du, Tianwei Lin, Wenhai Wang, et al. Planning-oriented autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3.1, 4.2.1

[38] Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. *arXiv preprint arXiv:2201.07207*, 2022. 2.2

[39] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre

Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608. 2.2

[40] Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer: Composable 3d value maps for robotic manipulation with language models. *arXiv preprint arXiv:2307.05973*, 2023. 2.2, 2.3.3

[41] Jyh-Jing Hwang, Runsheng Xu, Hubert Lin, Wei-Chih Hung, Jingwei Ji, Kristy Choi, Di Huang, Tong He, Paul Covington, Benjamin Sapp, et al. Emma: End-to-end multimodal model for autonomous driving. *arXiv preprint arXiv:2410.23262*, 2024. 4.2.1

[42] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. BC-Z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning (CoRL)*, 2022. 3.2

[43] Michael Janner, Yilun Du, Joshua B Tenenbaum, and Sergey Levine. Planning with diffusion for flexible behavior synthesis. *arXiv preprint arXiv:2205.09991*, 2022. 2.1, 2.4.1, 2.4.3, 3.2

[44] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. Motiondiffuser: Controllable multi-agent motion prediction using diffusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9644–9653, June 2023. 2.2

[45] Chiyu Jiang, Andre Cornman, Cheolho Park, Benjamin Sapp, Yin Zhou, and Dragomir Anguelov. MotionDiffuser: Controllable multi-agent motion prediction using diffusion. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 3.2

[46] Pushkal Katara, Zhou Xian, and Katerina Fragkiadaki. Gen2sim: Scaling up robot learning in simulation with generative models, 2023. 2.3.3

[47] Sanmin Kim, Hyeongseok Jeon, Jun Won Choi, and Dongsuk Kum. Diverse multiple trajectory prediction using a two-stage prediction network trained with lane loss. *IEEE Robotics and Automation Letters*, 8(4):2038–2045, apr 2023. doi: 10.1109/lra.2022.3231525. URL https://doi.org/10.1109/lra.2022.3231525. 2.2

[48] Seung Wook Kim, Jonah Philion, Antonio Torralba, and Sanja Fidler. DriveGAN: Towards a controllable high-quality neural simulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 3.2

[49] Alex Kuefler, Jeremy Morton, Tim Allan Wheeler, and Mykel John Kochenderfer. Imitating driver behavior with generative adversarial networks. *CoRR*,

abs/1701.06699, 2017. URL http://arxiv.org/abs/1701.06699. 2.2

[50] Kailin Li, Zhenxin Li, Shiyi Lan, Yuan Xie, Zhizhong Zhang, Jiayi Liu, Zuxuan Wu, Zhiding Yu, and Jose M Alvarez. Hydra-mdp++: Advancing end-to-end driving via expert-guided hydra-distillation. *arXiv preprint arXiv:2503.12820*, 2025. 4.2.1

[51] Zhenxin Li, Kailin Li, Shihao Li, Zhigang Ge, Jianqiang Wang, Yangjie Li, Haibing Luan, Jianming Hu, Zhengping Shen, and Hang Li. Hydra-MDP: End-to-end multimodal planning with multi-target hydra-distillation. *arXiv preprint arXiv:2406.06978*, 2024. 3.1

[52] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. *arXiv preprint arXiv:2209.07753*, 2022. 1, 2.2, 4.2.2

[53] Zhixuan Liang, Yao Mu, Mingyu Ding, Fei Ni, Masayoshi Tomizuka, and Ping Luo. Adaptdiffuser: Diffusion models as adaptive self-evolving planners. In *ICML*, 2023. 2.1, 3.2

[54] Bencheng Liao, Shaoyu Chen, Haoran Yin, Bo Jiang, Cheng Wang, Sixu Yan, Xinbang Zhang, Xiangyu Li, Ying Zhang, Qian Zhang, et al. Diffusiondrive: Truncated diffusion model for end-to-end autonomous driving. *arXiv preprint arXiv:2411.15139*, 2024. 4.2.1

[55] Bencheng Liao, Shaoyu Chen, Haoran Wang, Yanfeng Yin, and Xiangyu Zhang. DiffusionDrive: Truncated diffusion model for end-to-end autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 3.5.2, 3.5.2, 3.1

[56] Huihan Liu, Alice Chen, Yuke Zhu, Adith Swaminathan, Andrey Kolobov, and Ching-An Cheng. Interactive robot learning from verbal correction. *arXiv preprint arXiv:2310.17555*, 2023. 5.2

[57] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *CoRR*, abs/2107.13586, 2021. URL https://arxiv.org/abs/2107.13586. 2.2

[58] Cheng Lu, Huayu Chen, Jianfei Chen, Hang Su, Chongxuan Li, and Jun Zhu. Contrastive energy prediction for exact energy-guided diffusion sampling in offline reinforcement learning. *arXiv preprint arXiv:2304.12824*, 2023. 2.1

[59] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. In *Robotics: Science and Systems (RSS)*, 2021. 3.2

[60] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar.

Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023. 1, 4.2.2, 4.3.2

[61] Jiageng Mao, Yuxi Qian, Hang Zhao, and Yue Wang. Gpt-driver: Learning to drive with gpt. *arXiv preprint arXiv:2310.01415*, 2023. 2.2

[62] Ana-Maria Marcu, Long Chen, Jan Hünermann, Alice Karnsund, Benoit Hanotte, Prajwal Chidananda, Saurabh Nair, Vijay Badrinarayanan, Alex Kendall, Jamie Shotton, et al. Lingoqa: Visual question answering for autonomous driving. In *European Conference on Computer Vision*, pages 252–269. Springer, 2024. 4.2.3

[63] Oier Mees, Lukas Hermann, and Wolfram Burgard. What matters in language conditioned robotic imitation learning over unstructured data. In *IEEE Robotics and Automation Letters*, 2022. 3.2

[64] M Montremerlo, J Beeker, S Bhat, and H Dahlkamp. The stanford entry in the urban challenge. *Journal of Field Robotics*, 7(9):468–492, 2008. 2.2

[65] Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018. 3.2

[66] Nigamaa Nayakanti, Rami Al-Rfou, Aurick Zhou, Kratarth Goel, Khaled S. Refaat, and Benjamin Sapp. Wayformer: Motion forecasting via simple & efficient attention networks, 2022. URL https://arxiv.org/abs/2207.05844. 2.2

[67] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021. 2.2

[68] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 5.2

[69] Tim Pearce, Tabish Rashid, Anssi Kanervisto, Dave Bignell, Mingfei Sun, Raluca Georgescu, Sergio Valcarcel Macua, Shan Zheng Tan, Ida Momennejad, Katja Hofmann, and Sam Devlin. Imitating human behaviour with diffusion models, 2023. 2.1, 2.2

[70] Philip Polack, Florent Altché, Brigitte d'Andréa Novel, and Arnaud de La Fortelle. The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles? In *2017 IEEE intelligent vehicles symposium (IV)*, pages 812–818. IEEE, 2017. 2.4.1

[71] Dean A. Pomerleau. ALVINN: an autonomous land vehicle in a neural network.

In David S. Touretzky, editor, *Advances in Neural Information Processing Systems 1*, pages 305–313. San Francisco, CA: Morgan Kaufmann, 1989. 2.2

[72] Aditya Prakash, Kashyap Chitta, and Andreas Geiger. Multi-modal fusion transformer for end-to-end autonomous driving. *CoRR*, abs/2104.09224, 2021. URL https://arxiv.org/abs/2104.09224. 2.2, 3.5.2, 3.1

[73] Tianwen Qian, Jingjing Chen, Linhai Zhuo, Yang Jiao, and Yu-Gang Jiang. Nuscenes-qa: A multi-modal visual question answering benchmark for autonomous driving scenario. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 4542–4550, 2024. 4.2.3

[74] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36, 2024. 5.2

[75] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *ArXiv*, abs/2204.06125, 2022. 2.2

[76] Katrin Renz, Kashyap Chitta, Otniel-Bogdan Mercea, A. Sophia Koepke, Zeynep Akata, and Andreas Geiger. Plant: Explainable planning transformers via object-level representations. In *6th Annual Conference on Robot Learning*, 2022. URL https://openreview.net/forum?id=80vpxjt3vq. 2.1, 2.2, ??, 2.4.1

[77] Moritz Reuss, Maximilian Li, Xiaogang Jia, and Rudolf Lioutikov. Goal-conditioned imitation learning using score-based diffusion policies. *arXiv preprint arXiv:2304.02532*, 2023. 2.2

[78] Nicholas Rhinehart, Jeff He, Charles Packer, Matthew A Wright, Rowan McAllister, Joseph E Gonzalez, and Sergey Levine. Contingencies from observations: Tractable contingency planning with learned behavior models. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13663–13669. IEEE, 2021. 2.2

[79] Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685, 2022. 2.2

[80] Reuven Y Rubinstein. Optimization of computer simulation models with rare events. *European Journal of Operational Research*, 99(1):89–112, 1997. 2.1, ??, 2.4.3

[81] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Burcu Karagol Ayan, Seyedeh Sara

Mahdavi, Raphael Gontijo Lopes, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. *ArXiv*, abs/2205.11487, 2022. 2.2

[82] Oliver Scheel, Luca Bergamini, Maciej Wołczyk, Bla.zej Osi'nski, and Peter Ondruska. Urban driver: Learning to drive from real-world demonstrations using policy gradients. In *Conference on Robot Learning*, 2021. 2.1, 2.4.1, **??**

[83] Hao Sha, Yao Mu, Yuxuan Jiang, Li Chen, Chenfeng Xu, Ping Luo, Shengbo Eben Li, Masayoshi Tomizuka, Wei Zhan, and Mingyu Ding. Languagempc: Large language models as decision makers for autonomous driving. *arXiv preprint arXiv:2310.03026*, 2023. 2.2

[84] Nur Muhammad Shafiullah, Zichen Cui, Ariuntuya Arty Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *Advances in neural information processing systems*, 35:22955–22968, 2022. 2.2

[85] Hao Shao, Yuxuan Hu, Letian Wang, Guanglu Song, Steven L Waslander, Yu Liu, and Hongsheng Li. Lmdrive: Closed-loop end-to-end driving with large language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15120–15130, 2024. 4.2.1

[86] Lucy Xiaoyang Shi, Zheyuan Hu, Tony Z Zhao, Archit Sharma, Karl Pertsch, Jianlan Luo, Sergey Levine, and Chelsea Finn. Yell at your robot: Improving on-the-fly from language corrections. *arXiv preprint arXiv:2403.12910*, 2024. 5.2

[87] Chonghao Sima, Katrin Renz, Kashyap Chitta, Li Chen, Hanxue Zhang, Chengen Xie, Jens Beißwenger, Ping Luo, Andreas Geiger, and Hongyang Li. Drivelm: Driving with graph visual question answering. In *European Conference on Computer Vision*, pages 256–274. Springer, 2024. 4.2.3

[88] Jascha Narain Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. *ArXiv*, abs/1503.03585, 2015. 2.2

[89] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020. 2.2, 2.3.1

[90] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2446–2454, 2020. 1, 4.2.3

[91] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning, 2023. 2.2

[92] Shuhan Tan, Boris Ivanovic, Yuxiao Chen, Boyi Li, Xinshuo Weng, Yulong Cao, Philipp Krähenbühl, and Marco Pavone. Promptable closed-loop traffic simulation. *arXiv preprint arXiv:2409.05863*, 2024. 4.2.3

[93] Yichuan Charlie Tang and Ruslan Salakhutdinov. Multiple futures prediction. *CoRR*, abs/1911.00997, 2019. URL http://arxiv.org/abs/1911.00997. 2.2

[94] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. 3.2

[95] Julen Urain, Niklas Funk, Georgia Chalvatzaki, and Jan Peters. Se (3)-diffusionfields: Learning cost functions for joint grasp and motion optimization through diffusion. *ICRA*, 2023. 2.1

[96] Xiaofei Wang, Kimin Lee, Kourosh Hakhamaneshi, Pieter Abbeel, and Michael Laskin. Skill preferences: Learning to extract and execute robotic skills from human feedback. In *Conference on Robot Learning*, pages 1259–1268. PMLR, 2022. 5.2

[97] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. Diffusion policies as an expressive policy class for offline reinforcement learning. *arXiv preprint arXiv:2208.06193*, 2022. 2.1

[98] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2022. URL https://arxiv.org/abs/2201.11903. 2.2

[99] Licheng Wen, Xuemeng Yang, Daocheng Fu, Xiaofeng Wang, Pinlong Cai, Xin Li, Tao Ma, Yingxuan Li, Linran Xu, Dengke Shang, et al. On the road with gpt-4v (ision): Early explorations of visual-language model on autonomous driving. *arXiv preprint arXiv:2311.05332*, 2023. 2.2

[100] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015. 2.1, 2.3.2, 2.4.3

[101] Zhou Xian, Nikolaos Gkanatsios, Théophile Gervet, Tsung-Wei Ke, and Katerina Fragkiadaki. Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation. In *CoRL 2023*. 2.2

[102] Shuo Xing, Chengyuan Qian, Yuping Wang, Hongyuan Hua, Kexin Tian, Yang Zhou, and Zhengzhong Tu. Openemma: Open-source multimodal model for end-to-end autonomous driving. In *Proceedings of the Winter Conference on Applications of Computer Vision*, pages 1001–1009, 2025. 4.2.1

[103] Danfei Xu, Roberto Martín-Martín, De-An Huang, Yuke Zhu, Silvio Savarese, and Li Fei-Fei. Regression planning networks. *CoRR*, abs/1909.13072, 2019. URL http://arxiv.org/abs/1909.13072. 2.2

[104] Zhenhua Xu, Yujia Zhang, Enze Xie, Zhen Zhao, Yong Guo, Kenneth KY Wong, Zhenguo Li, and Hengshuang Zhao. Drivegpt4: Interpretable end-to-end autonomous driving via large language model. *arXiv preprint arXiv:2310.01412*, 2023. 2.2

[105] Mengjiao Yang, Yilun Du, Kamyar Ghasemipour, Jonathan Tompson, Dale Schuurmans, and Pieter Abbeel. Learning interactive real-world simulators. *ArXiv*, abs/2310.06114, 2023. 2.2

[106] Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas, Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, et al. Language to rewards for robotic skill synthesis. *arXiv preprint arXiv:2306.08647*, 2023. 2.3.3, 3.2, 4.2.2

[107] Wenyuan Zeng, Wenjie Luo, Simon Suo, Abbas Sadat, Bin Yang, Sergio Casas, and Raquel Urtasun. End-to-end interpretable neural motion planner. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8660–8669, 2019. 2.2

[108] Zhejun Zhang, Alexander Liniger, Dengxin Dai, Fisher Yu, and Luc Van Gool. End-to-end urban driving by imitating a reinforcement learning coach. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. 2.2

[109] Ziyuan Zhong, Davis Rempe, Danfei Xu, Yuxiao Chen, Sushant Veer, Tong Che, Baishakhi Ray, and Marco Pavone. Guided conditional diffusion for controllable traffic simulation. *ArXiv*, abs/2210.17366, 2022. 2.2, 3.2

[110] Xingcheng Zhou, Xuyuan Han, Feng Yang, Yunpu Ma, and Alois C Knoll. Opendrivevla: Towards end-to-end autonomous driving with large vision language action model. *arXiv preprint arXiv:2503.23463*, 2025. 4.2.1

[111] Yifeng Zhu, Jonathan Tremblay, Stan Birchfield, and Yuke Zhu. Hierarchical planning for long-horizon manipulation with geometric and symbolic scene graphs. *CoRR*, abs/2012.07277, 2020. URL https://arxiv.org/abs/2012.07277. 2.2

[112] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008. 3.2

[113] Julius Ziegler, Philipp Bender, Thao Dang, and Christoph Stiller. Trajectory planning for bertha—a local, continuous method. In *2014 IEEE intelligent vehicles symposium proceedings*, pages 450–457. IEEE, 2014. 2.2