Autonomous Vehicle Path Planning by Searching With Differentiable Simulation

Asen Nachkov[1], **Jan-Nico Zaech**[1], **Danda Pani Paudel**[1], **Xi Wang**[2], **Luc Van Gool**[1]

[1]INSAIT, Sofia University "St. Kliment Ohridski", Sofia, Bulgaria
[2]ETH Zurich, Zurich, Switzerland

Abstract

Planning allows an agent to safely refine its actions before executing them in the real world. In autonomous driving, this is crucial to avoid collisions and navigate in complex, dense traffic scenarios. One way to plan is to search for the best action sequence. However, this is challenging when all necessary components – policy, next-state predictor, and critic – have to be learned. Here we propose Differentiable Simulation for Search (DSS), a framework that leverages the differentiable simulator Waymax as both a next state predictor and a critic. It relies on the simulator's hardcoded dynamics, making state predictions highly accurate, while utilizing the simulator's differentiability to effectively search across action sequences. Our DSS agent optimizes its actions using gradient descent over imagined future trajectories. We show experimentally that DSS – the combination of planning gradients and stochastic search – significantly improves tracking and path planning accuracy compared to sequence prediction, imitation learning, model-free RL, and other planning methods.

1 Introduction

When a human driver notices an approaching vehicle in the rear-view mirror, they expect to see the overtaking vehicle in front of them in the next several seconds. This intuitive subconscious anticipation is a form of world modeling and enables planning – the process of selecting the right actions by predicting and assessing their likely effects. For a computational driving agent, planning is also crucial in order to drive safely and reliably across diverse scenes and conditions.

One way to perform planning is to *search* for the best action sequence across multiple candidates. The planning agent imagines a number of them, uses its world model to estimate their effects, rates them according to preferences and task constraints, and selects the best one. This is intuitive, yet practical questions are still plentiful. For example, should the agent consider more trajectories or focus and refine a few of them? In this context, we demonstrate that *differentiable simulation* is well-suited for this search problem and enables very efficient planning, as we show below.

Generally, developing accurate search capabilities is not easy. Intuitively, one needs three modules: a policy to suggest actions, a state predictor to predict their effect, and a critic to score them. All modules need to be accurate so the agent can both propose good actions and recognize them as being good. Previous works have attained impressive results in controlled symbolic environments where only some of these modules have to be *learned* (Silver et al. 2016; Moura and Ullrich 2021). In fact, the general insight is that environments where we can obtain such components nearly optimally are more suitable for successful planning. This is because a hardcoded realistic simulator used as a state predictor, or a symbolic engine used as a critic, is almost always more accurate than their learned counterparts, facilitating searching.

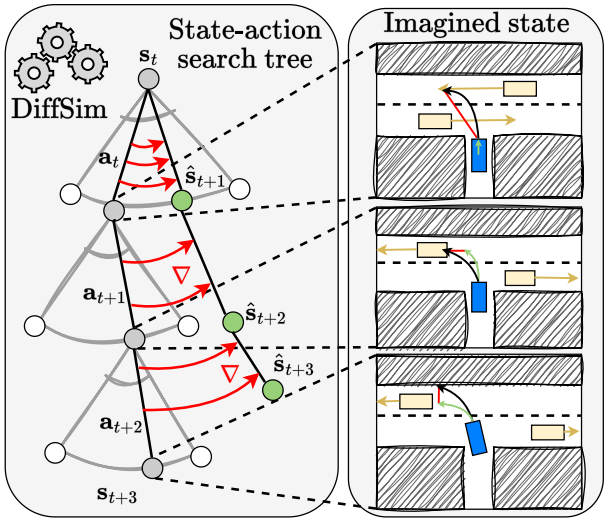

Figure 1: **Differentiable simulation at test time**. To select the current action, the agent uses a differentiable simulator to imagine a future trajectory ($\mathbf{s}_t, ..., \mathbf{s}_{t+3}$, gray circles), sampled from a distribution of possible states (arcs, white circles). The imagined trajectory is refined using gradient descent towards an optimal one ($\hat{\mathbf{s}}_{t+1}, ..., \hat{\mathbf{s}}_{t+3}$, green circles).

In the field of autonomous vehicles, test-time search has been underexplored, likely due to challenges like continuous action spaces, diverse realistic scenes, and limited data scale. To search efficiently, we require (i) the next-state predictor and critic to be as accurate as possible, so the agent's imagination faithfully represents a realistic probable future, and (ii) the relationship between the agent's actions and the imagined outcome to be "easy-to-model", so that a small change in

the actions leads to a small change in the imagined outcome. To satisfy these we use the differentiable simulator Waymax (Gulino et al. 2024) as both a next-state predictor and a critic. Its hardcoded dynamics are realistic, making any sampled Monte Carlo trajectory highly informative. By being differentiable, we can backpropagate through it and any error in the imagined outcome induces a proportional error in the agent's imagined actions. Overall, while a simulator itself provides only evaluative feedback, like in classic reinforcement learning (Sutton, Barto et al. 1998), a differentiable simulator, as used by us, provides instructive feedback through the loss gradients, pointing to the direction of maximal increase and allowing the agent to more efficiently search through the possible action sequences.

Our approach to test-time planning, shown in Fig. 1, is called **Differentiable Simulation for Search** (DSS). The agent imagines a future trajectory in which it acts according to a trained policy. The trajectory is *virtual* and is obtained in an autoregressive manner (step by step) using the simulator. Subsequently, the simulator rates the trajectory, based on for example whether there are any collisions or offroad events, and a planning loss is computed. The gradients of this loss propagate through the differentiable dynamics in a manner similar to backpropagation-through-time (BPTT) and reach the ego-vehicle's actions, updating them towards an optimum. The optimized actions can then be executed in the real world, thereby forming a real *physical* trajectory.

Our DSS approach is designed for single-agent control. Within the virtual future other agents should evolve according to how the ego-vehicle imagines them. Thus, to obtain their future expected locations, the ego-vehicle needs to perform multiagent control. From its own perspective this is natural, given that everything is controllable within one's own imagination. From the perspective of Waymax, however, it requires controlling all agents using a learned policy. Waymax works by replaying the historical real-world scenarios from the Waymo Open Motion Dataset (WOMD) (Ettinger et al. 2021). Those agents not controlled by a learned, user-provided policy are evolved according to their historic motion, which the ego-vehicle is not supposed to see, creating a potential data leakage. Hence, within the virtual future, all agents have to be controlled.

To run such a planner in a real vehicle, one would need a sensing stack with (i) state *estimation* software that initializes the simulator state from real sensory data (surrounding cameras, LiDAR, IMUs), keeps track of appearing and disappearing objects, receives navigation, and (ii) state *simulation* software, which steps the dynamics forward and generates the virtual trajectories. While Waymax only provides state simulation, we sidestep the need to build state estimation ourselves by adopting the WOMD scenarios and their 9 second episodes, allowing us to study the planning task in isolation.

The Waymax simulator provides differentiable dynamics mapping state-action pairs to next states, which in DSS become the next-state predictor. It also provides calculations like collision and offroad detection, which in our framework become the critic and have to be used in the planning loss function. Yet, they are boolean and non-differentiable. To be able to extract instructive feedback from them, they have to be differentiable. Thus, we propose **Classifier-Guided Action Selection** – a simple approach that uses a small learnable classifier, implemented as a neural network, to detect any undesirable non-differentiable events, such as offroad or collisions. Similar to how classifier-guided diffusion is used to generate images (Dhariwal and Nichol 2021), this technique is used to steer actions away from collision and offroad events, according to their likelihoods from the classifier.

This design renders our DSS framework efficient due to the use of a differentiable simulator, realistic owing to the planning, and inherently interpretable. It guarantees that any actions executed are chosen based on per-scenario optimization, going beyond reactive decision-making. It achieves 16 times lower displacement error than a reactive baseline at tracking, is 2 times better at realistic path planning, and obtains almost 2 times better collision rates compared to state-of-the-art trackers. Overall, our contributions are threefold:

1. We propose DSS – a framework that allows searching for the best action-sequence within a continuous action differentiable simulator at test time, in Sec. 3.2.
2. We propose Classifier-Guided Action Selection as a simple way to approximate non-differentiable events and to plan against them, in Sec. 3.3.
3. We implement and evaluate the framework for both tracking and path planning, showing its benefits compared to state-of-the-art methods and baselines.

2 Related Work & Context

Our work stands at the intersection of differentiable simulation (DiffSim) and classic RL planning methods.

DiffSim for driving. Differentiable coupling of different modules is not new within autonomous vehicles. TrAAD (Zheng, Son, and Lin 2022) and TrafficSim (Suo et al. 2021) have modeled traffic interactions with differentiable ODE-based car-following dynamics or an entirely learned multi-agent model, respectively. DIPP (Huang et al. 2023) integrates a differentiable planner, allowing joint optimization of motion prediction and planning objectives. Compared to this, our DSS framework updates the trajectory using gradient descent instead of solving a full optimization problem, and can refine the actions based on multiple likely futures. DiffStack (Karkus et al. 2023) composes a fully differentiable AV stack where gradients flow through learned modules like differentiable MPC. Recently, APG (Nachkov, Paudel, and Van Gool 2025) has focused on differentiable dynamics, as opposed to author-designed differentiable modules. There, DiffSim makes the policy rollout differentiable, allowing end-to-end training, on which also our framework relies.

DiffSim in robotics. Within robotics, differentiable simulation is rapidly becoming popular. It allows estimating physical object properties from simulation (Geilinger et al. 2020) and training robotic policies (Lutter et al. 2021; Toussaint et al. 2018; Qiao et al. 2020; Holl, Koltun, and Thuerey 2020). Differentiable contact models have been used for object manipulation (Li et al. 2023; Xu et al. 2021, 2023; Lin et al. 2022). APG has found applications in aerial navigation with fixed-wing drones (Wiedemann et al. 2023), quadruped locomotion (Song, Kim, and Scaramuzza 2024), and for quadrotor

Method	Policy	Next state	Critic	Action selection
PG	L	—	—	Reactive
APG	L	—	—	Reactive
Dyna	L	L	L	Reactive
MPC	—	L/S	L	MPC
AWM-MPC	L	L	L	MPC
AlphaGo	L	S	L	MCTS
MuZero	L	L	L	MCTS
AlphaGeometry	L	SE	L	BS
AlphaProof	L	PS	L	MCTS
AlphaTensor	L	S	L	MCTS
LLM reasoning	L	—	L	BS
DSS (ours)	L	S	S + L	MPC + ∇

Table 1: **Relevant method characteristics**. DSS (ours) is the only one that uses differentiable simulation at test time. L = learned neural net, S = simulator, SE = symbolic engine, PS = proof system, MCTS = Monte Carlo tree search, BS = beam search, MPC = model predictive control, ∇ = gradients.

control from visual features (Heeg, Song, and Scaramuzza 2024). However, in contrast to our approach, in all these applications DiffSim is only used during training, and the policy is simply rolled out at test time.

Planning. In RL, many promising models have used planning in diverse environments, attaining strong results. We list them in Table 1, highlighting their *main* differences. Methods like policy gradients (PG) (Sutton et al. 1999) or DQN (Mnih et al. 2015) are reactive in nature since they act by following a learned explicit or implicit policy. APG follows them but uses DiffSim at training time. Dyna-style algorithms (Sutton 1991; Hafner et al. 2019) use a world model to train on imagined transitions but are reactive at test time. MPC uses a world model to predict trajectories, rate them, and select the best one. Variants based on DiffSim have also been developed (Nachkov et al. 2026). For board games, AlphaGo (Silver et al. 2016) and MuZero (Schrittwieser et al. 2020) use sophisticated planning in discrete action spaces, while methods like AlphaGeometry (Trinh et al. 2024), AlphaProof (AlphaProof and AlphaGeometry teams 2024), and AlphaTensor (Fawzi et al. 2022) rely on symbolic engines or proof systems to facilitate the search. Recently, LLMs have been used as policies and been combined with test-time search strategies such as Best-of-N or beam search (Snell et al. 2024). There, a process reward model acts as a learned critic to score answers (Luo et al. 2024). Compared to all these methods, our DSS planning framework is the only one that uses a differentiable simulator at test time to search for the right action. Like them, we use a learned policy that captures the right statistical patterns from the scenarios at training time.

State-of-the-art in Waymax. Within the Waymax setting, there are hardly any baselines that use planning. We compare against behavior cloning (Gulino et al. 2024) and APG approaches, sequence prediction approaches such as Wayformer (Nayakanti et al. 2023), offline RL methods like Decision Transformer (DT) (Chen et al. 2021) and more recent Waymax state-of-the-art RL baselines like EasyChauffeur (Xiao et al. 2024) and PiDT (Zhou et al. 2025). None of these are exact alternatives to our planning framework because they are designed to rely on privileged information. For example, EasyChaffeur uses the full historic trajectory as a route to condition the agent (more akin to path tracking, rather than autonomous navigation), whereas our setup is more realistic and only uses the last (x, y) waypoint to indicate a final destination. Additionally, previous methods have different training objectives, often aiming to reproduce historical expert actions, whereas our training aims to reproduce historical expert states – a subtle nuance.

3 Method

Notation. We represent the set of all states as $\mathcal{S}$ and that of the actions as $\mathcal{A}$. The simulator is abstracted as a pure, stateless differentiable function, $\text{Sim} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$, that maps state-action pairs to next states, $\text{Sim}(\mathbf{s}_t, \mathbf{a}_t) \mapsto \mathbf{s}_{t+1}$. A trajectory is a sequence of state-action pairs $(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T)$. We can extract agent locations $(\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_T)$ and action sequences $(\mathbf{a}_0, ..., \mathbf{a}_{T-1})$ from it. We denote an action of the $e$go-vehicle as $\mathbf{a}_t^e$, and an action of all $o$ther agents as $\mathbf{a}_t^o$. Actions are vector-valued with a dimension A and represent acceleration and steering in our setting.

DSS agent. Our DSS framework requires a learned stochastic policy to model agent behavior. Its training is described in Sec. 3.1, and our approach to planning at test time in Sec. 3.2. Classifier-Guided Action Selection, in Sec. 3.3, allows us to model non-differentiable events when planning.

3.1 Training – Analytic Policy Gradients

We need to learn a stochastic policy π_θ for producing the reactive behavior of the agents. We train it using Analytic Policy Gradients (APG) (Nachkov, Paudel, and Van Gool 2025) to learn a realistic action distribution from historical expert driver trajectories. Specifically, we train on the WOMD scenarios within Waymax. In each scenario the agent performs a rollout, after which the full obtained trajectory $(\mathbf{s}_0, ..., \mathbf{s}_T)$ is supervised with the expert human driver one, $(\hat{\mathbf{s}}_0, ..., \hat{\mathbf{s}}_T)$. Gradients flow through the dynamics, similar to BPTT:

$$\begin{aligned} \min_\theta \ \mathcal{L}_{\text{train}} &= \|\mathbf{s}_t - \hat{\mathbf{s}}_t\|_2^2 \\ \text{with } \mathbf{s}_t &= \text{Sim}(\mathbf{s}_{t-1}, \pi_\theta(\mathbf{s}_{t-1})). \end{aligned} \tag{1}$$

Action selection. The policy π_θ is stochastic and is parametrized as a Gaussian mixture with six components. To encourage action multimodality during training, actions for the rollout are selected by sampling not from the entire Gaussian mixture, but only from that one component that will bring the ego-vehicle closest to the next expert state (Nayakanti et al. 2023). We use the simulator to find that component efficiently (details in the suppl. materials). The error signals during backpropagation reach only this component, instead of all of them. This allows the policy to sample diverse actions, which is beneficial for searching at test time.

Recurrent architecture. Since the policy is recurrent, its hidden state encapsulates the entire history of observations, represented as $\pi_\theta(\mathbf{a}_t|\mathbf{s}_{\leq t})$. For computational efficiency during training we only control the ego-vehicle, while the other

Algorithm 1: DSS

Input: Initial state $\mathbf{s}_0$, policy π_θ, simulator Sim, loss function ℓ, imagination horizon T, number of rollouts K, discount factor γ, temperature τ, step size η

Output: Ego action $\mathbf{a}_0$ to apply at initial state $\mathbf{s}_0$

1 **for** $k \leftarrow 0$ **to** $K-1$ **do**
2 Extract agent states $\mathbf{x}_0^{e,k}, \mathbf{x}_0^{o,k} \leftarrow \mathbf{s}_0$
3 **for** $t \leftarrow 0$ **to** $T-1$ **do**
4 $\mathbf{a}_t^{e,k}, \mathbf{a}_t^{o,k} \leftarrow \pi_\theta(\mathbf{x}_{\le t}^{e,k}, \mathbf{x}_{\le t}^{o,k})$
5 $\mathbf{x}_{t+1}^{e,k}, \mathbf{x}_{t+1}^{o,k} \leftarrow \mathrm{Sim}(\mathbf{x}_t^{e,k}, \mathbf{x}_t^{o,k}, \mathbf{a}_t^{e,k}, \mathbf{a}_t^{o,k})$
6 Compute rollout losses $\ell^k = \ell(\mathbf{x}_{1:T}^{e,k})$ for all k
7 Compute ego action gradients $\mathbf{g}_0^k = \partial \ell^k / \partial \mathbf{a}_0^{e,k}$
8 Compute weights:
$w_k = \big(\exp(-\ell^k/\tau)\big) / \big(\sum_{j=0}^{K-1} \exp(-\ell^j/\tau)\big)$
9 **return** $\sum_{k=0}^{K-1} w_k \big(\mathbf{a}_0^{e,k} - \eta \mathbf{g}_0^k\big)$

agents' states evolve according to their historic motion. However, at test time the policy could be used to control also the other agents. By extracting state observations from the perspective of all agents we can compute all actions in parallel. We overwrite the notation as $\mathbf{a}_t^e, \mathbf{a}_t^o = \pi_\theta(\mathbf{x}_{\le t}^e, \mathbf{x}_{\le t}^o)$, where $\mathbf{x}_t^e$ and $\mathbf{x}_t^o$ indicate the $e$go and $o$ther agents' positions at time t. Full implementation details are in the suppl. materials.

3.2 Testing – Differentiable Simulation for Search

Our planning algorithm at test time, called Differentiable Simulation for Search (DSS), is shown in Algorithm 1. To select the current action, the ego-agent *imagines* K future trajectories, each of length T steps. They are generated autoregressively by using the trained policy π_θ to compute actions for both the ego- and the other agents, while the simulator is used to compute their next states, in lines 3–5. Having obtained the K trajectories, we compute a loss function over the ego positions, line 6. With DiffSim, in lines 7 and 9 we can compute the gradients of the loss with respect to the first ego-action and perform a single gradient descent step to improve it. Since the algorithm is based on sampled imaginary rollouts, the final selected action is a weighted average of the optimized first actions from these rollouts, where actions in trajectories with lower losses have higher weights. Fig. 2 shows how the gradients improve the trajectory.

Flexibility. Alg. 1 encompasses a full set of possible behaviors for how the agent can plan its action at test time (see Fig. 3). The setting $K = 1, T = 1, \eta = 0$ represents a **reactive agent** that drives by relying on the trained policy π_θ. If $K = 1$ but $\eta > 0$, the agent uses the differentiability of the simulator to optimize its actions, as the gradient step size η is positive, but does not use the simulator to perform Monte Carlo search. We call this setting **reactive with gradients**. If $K > 1$ and $\eta = 0$ the agent uses multiple Monte Carlo rollouts to search for the right actions but does not use gradient descent to optimize them. This setting is called **simulator as a critic**, because the simulator computes trajectories and losses, but its differentiability is not used. Finally, our full proposed setting **differentiable simulator as a critic** is enabled when gradients are used to optimize the actions, i.e. $\eta > 0$, and multiple rollouts are used to search, i.e. $K > 1$.

Algorithm 2: Main Control Loop

Input: Initial state $\mathbf{s}_0$, rollout horizon T, rollouts K, actions to execute M

Output: Continuous sequence of executed ego actions

1 $\mathbf{s} \leftarrow \mathbf{s}_0$
2 **while** $\mathbf{s}$ *not final* **do**
3 $\mathbf{a}_{0:M-1} \leftarrow \mathrm{Plan}(\mathbf{s}, K, T, M)$ # Algorithm 1
4 **for** $i \leftarrow 0$ **to** $M-1$ **do**
5 $\mathbf{s}_{t+1} \leftarrow \mathrm{Sim}(\mathbf{s}_t, \mathbf{a}_t)$

Control loop. The imagined trajectories in Alg. 1 are virtual – they represent the future as predicted by the ego-agent's policy. Algorithm 2 provides the main loop for obtaining a real, physical trajectory, over which the evaluation metrics are calculated. Specifically, instead of planning out only the first action, we optimize and execute the first M imagined actions, after which the ego-vehicle has to re-plan. In line 5, only the ego-agent is controlled by the policy's optimized actions. This is in contrast to line 5 in Alg. 1 where all agents are controlled. The overall effect is that Alg. 1 shows *how the ego-vehicle optimizes its own actions within the virtual, imagined dynamics, which inherently involves multi-agent control in order to imagine the other agents' motions*, while Alg. 2 is used to obtain the real physical trajectories, where only the ego-vehicle is controlled by the policy π_θ.

Computational cost. When re-planning once every M steps, the reaction time to any observation can be up to M steps. Re-planning once every 3 steps corresponds to a reaction time of at most 0.3 seconds (at 10 frames per second). The total computational cost, in number of policy calls, is $O(LKTN/M)$ where L is the length of the physical trajectory, and N is the maximum number of actors in the scene. For Waymax, where $L = 90$ and $N = 128$, and when $M = 3, K = 8, T = 10$, this runs at 4.1 seconds per scenario on a single RTX3090 GPU. Thus, a full scenario long 90 timesteps, or 9 seconds of historical real driving, is processed in 4.1 seconds – effectively *real-time*.

3.3 Classifier-Guided Action Selection

The function ℓ in line 6 in Alg. 1 is the main objective to optimize when planning. It should contain loss terms for any undesirable behavior, while not leaking any privileged information such as future ground-truth locations. The **offroad** and **collision** functions in Waymax are of interest, yet are boolean and non-differentiable. Previous works (Nachkov et al. 2026) have used 2D Gaussians to approximate the rotated boxes, which yields a closed-form, differentiable overlap formula, but is limited, as it applies only to collisions.

Inspired by classifier-guidance for image generation, we propose Classifier-Guided Action Selection. We approximate the non-differentiable collision and offroad detection func-

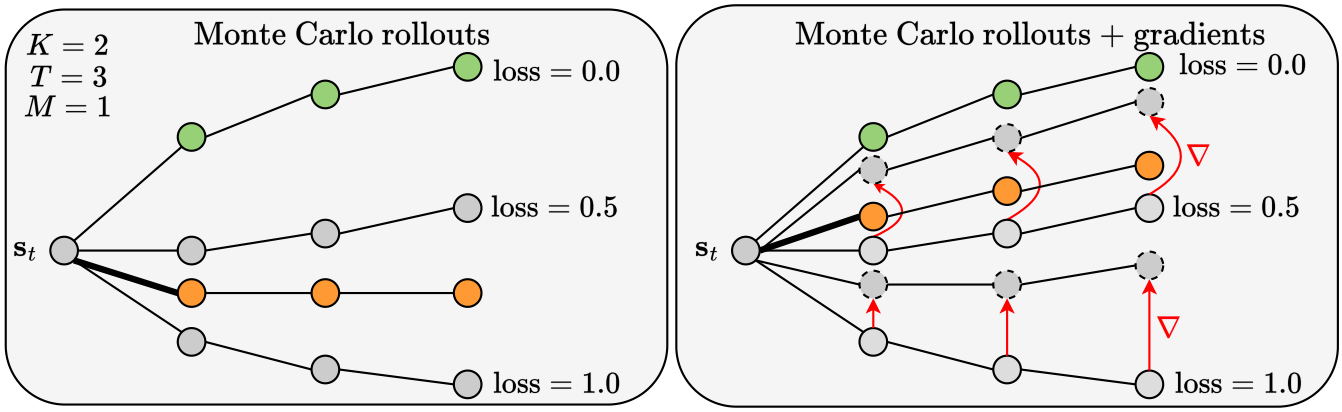

Figure 2: **Gradient descent in the ego-agent's imagined future**. Without gradients (left), searching involves sampling K trajectories (gray) of length T, scoring them, and aggregating their first M actions (**bold** black line). The trajectory from the resulting actions is shown in orange. With gradients, each rolled out trajectory is first updated towards an optimum (green), as judged by the planning loss. The executed trajectory from the aggregated actions more closely aligns with this optimal trajectory.

tions with a simple multi-label classifier, p_ϕ, trained on simulated trajectories. We indicate the collision and offroad events as c and o, respectively. During training the simulator computes the ground-truth binary labels M_c and M_o for whether such events are present. Then, in Eqn. 2 we formulate the collision and offroad losses – negative binary cross-entropy – with which the classifier is trained.

$$\ell_t^c = -\Big(M_c \log p_\phi(c|\mathbf{s}_t) + (1 - M_c) \log \big(1 - p_\phi(c|\mathbf{s}_t)\big)\Big)$$
$$\ell_t^o = -\Big(M_o \log p_\phi(o|\mathbf{s}_t) + (1 - M_o) \log \big(1 - p_\phi(o|\mathbf{s}_t)\big)\Big) \tag{2}$$

The full loss to minimize during planning is the average collision and offroad probability across all imagined steps:

$$\min_{\mathbf{a}_t,\dots,\mathbf{a}_{t+T}} \ell = \frac{1}{T}\sum_{t=0}^{T-1}\Big(M_c p_\phi(c|\mathbf{s}_t) + M_o p_\phi(o|\mathbf{s}_t)\Big). \tag{3}$$

The gradient $\partial\ell/\partial\mathbf{a}_t^e$ goes through the classifier p_ϕ, whose weights ϕ are frozen, through the state $\mathbf{s}_{t+1}$, the differentiable dynamics $\text{Sim}(\mathbf{s}_t, \mathbf{a}_t)$, and reaches the actions $\mathbf{a}_t$. In this way, thanks to differentiable simulation, the ego-agent can optimize its actions at test time. Importantly, even if some desirable components are non-differentiable, they can still be optimized through search, e.g. when $\eta = 0$ and $K > 1$. Hence, our DSS framework is flexible in handling non-differentiable losses, in which case it simply becomes loss-weighted Monte Carlo search over the sampled trajectories.

4 Experiments

In this section we validate our proposed approach experimentally. The two research questions we seek to answer are:

- *Is our proposed search procedure beneficial in general?* We assess this in Sec. 4.1 by adopting a well-behaved planning loss function that models a tracking problem.
- *Is it beneficial specifically for AV path planning?* For this we adopt a realistic, restrictive planning loss function in Sec. 4.2 that models a difficult path planning problem.

Experimental setup. The **inputs** to the ego-agent's policy π_θ include the locations of all agents, the nearest roadgraph points, the traffic lights, the agent's own speed, and the last (x, y) waypoint from the expert trajectory, which is needed to mark the final destination (otherwise, without an intended destination one cannot expect to compare to the expert trajectory). The **outputs** are actions – acceleration and steering.

We follow the evaluation protocol in previous works (Nachkov, Paudel, and Van Gool 2025) and use the Waymax simulator, which builds over the WOMD scenarios. For each scenario we compute the **displacement error** (= ADE = L_2 distance) of the ego-vehicle's physical trajectory compared to the historic one. The number reported is averaged over the timesteps within the trajectory and over all scenarios in the validation set. Further, we track the **overlap** and **offroad rates**. They indicate the proportion of scenarios in which at least one ego collision or offroad event occurs.

4.1 Evaluating the Search Framework

Here we evaluate the general DSS framework presented in Sec. 3.2. The planning loss function, line 6 in Alg. 1, is the L_2 distance between the simulated and the expert trajectory:

$$\ell = \frac{1}{T}\sum_{t=0}^{T-1}\|\mathbf{x}_t^e - \hat{\mathbf{x}}_t^e\|_2^2 \tag{4}$$

This setting represents a tracking problem – given a path of time-dependent waypoints $\hat{\mathbf{x}}_0^e, \dots, \hat{\mathbf{x}}_{T-1}^e$, optimize for the actions that follow it. The best action for the next timestep necessarily belongs to the sequence of best actions for all future timesteps. Hence, the agent can afford to be greedy and not plan very far ahead into the future (we can set $T = 1$).

Evaluations of different settings. Importantly, we visualize the logical relationships between the different framework settings as an ablation tree, shown in Fig. 3. Depending on whether gradient updates are used ($\eta > 0$), and whether searching is used ($K > 1$), four configurations are available.

The **reactive** settings are shown in Table 2. Results are deterministic because when $K = 1$, the agent only imagines a single trajectory, which we take to be formed by always

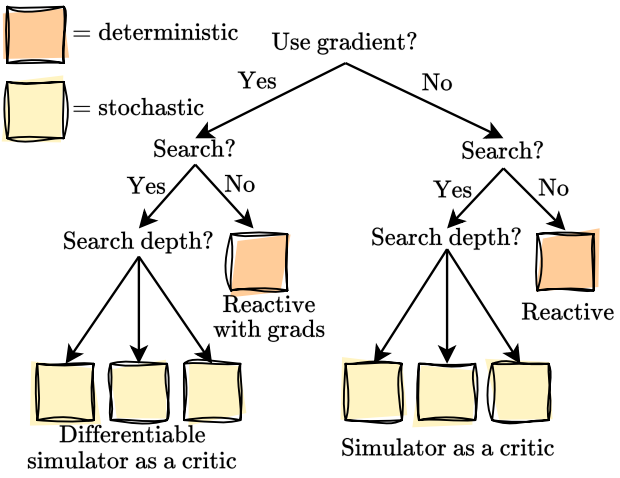

Figure 3: **Experiment ablation tree**. Depending on the different configuration of whether to search and use gradients, there are four different experimental settings. Our full framework DSS uses both search across multiple trajectories and gradients to optimize the actions across them.

η_{accel}	η_{steer}	ADE ↓	overlap ↓	offroad ↓
0	0	2.9792	0.1369	0.0780
100	0.01	2.1294	0.0816	0.0718
200	0.01	1.5282	0.0558	0.0686
500	0.01	0.7193	0.0327	0.0660
1000	0.01	**0.4644**	**0.0269**	0.0664
5000	0.01	0.7678	0.0311	**0.0653**

Table 2: **Effect of the gradient step size in reactive settings.** Here we set $T = 1$. The best results require a large step size for the acceleration and small one for the steering. *Takeaway*: guiding the reactive policy using the gradient could greatly improve performance, even when not searching.

selecting the mean action (there is no reason to choose otherwise). Since performance is more sensitive to wrong steering than to wrong acceleration, we use different learning rates for the action elements. A high learning rate for the acceleration (1000) is beneficial, yet a too high one (5000) starts causing the update to overshoot the optimal action.

In the **simulator as a critic** setting we perform search when $K > 1$ without gradient optimization. Table 3 shows that results improve as more trajectories are imagined. By imagining different likely future sequences, the agent can find, *by chance*, more accurate action sequences.

Rollouts K	ADE ↓	overlap ↓	offroad ↓
1	2.9792	0.1369	0.0780
2	1.1954	0.0381	0.0663
4	1.0968	**0.0369**	0.0653
8	**1.0710**	0.0378	**0.0646**

Table 3: **Simulator as predictor and critic.** We sample K trajectories from a stochastic policy with $T = 1, \eta = 0, \tau = 1$. *Takeaway*: using the simulator to score the Monte Carlo trajectories and to search through them yields strong results.

Finally, Table 4 shows the same setting but with gradient updates enabled in addition to the search, this being the full proposed **differentiable simulator as a critic** setting. The agent can now both find good actions by chance, and further optimize them through the differentiable simulator. By setting the sampling temperature τ to be low, the agent can select the best action. Note that in some cases selecting the maximum from multiple noisy values, as done for example in DQNs (Mnih et al. 2015), may introduce instability and hurt performance. Here, since the simulator is a perfect critic, we avoid this issue. The standard deviations for the 3 metrics over 5 random seeds are $(0.01, 0.0008, 0.0007)$ and statistical variability plays almost no role in our results.

Model	ADE ↓	overlap ↓	offroad ↓
DecisionTransformer	8.3200	0.0362	0.0621
BC	3.6000	0.1120	0.1359
PiDT	6.9900	0.0186	0.0298
Wayformer	2.3800	0.1068	0.0789
EasyChaffeur-IL	—	0.0293	0.0280
EasyChaffeur-PPO	—	0.0443	**0.0216**
Ours ($K = 1$)	0.4644	0.0269	0.0664
Ours ($K = 2$)	0.2509	0.0196	0.0498
Ours ($K = 4$)	0.2013	0.0168	0.0414
Ours ($K = 8$)	**0.1766**	**0.0150**	0.0350

Table 4: **DSS for trajectory tracking.** Here $T = 1, \tau = 0.01$ and the learning rate is $\eta = (1000, 0.01)$. *Takeaway*: the search allows the agent to find good actions by chance, while the gradients further refine them.

Discussion. The results validate the benefits of our DSS procedure. For tracking a log-trajectory the searching and gradient updates improve over the reactive performance by up to 16.9 times (2.979 → 0.176 ADE). This occurs because (i) the Monte Carlo searching against the simulator provides very accurate information about the trajectory's quality, and (ii) the simulator is differentiable and we can update the actions to minimize the loss function. Compared to other methods – behavior cloning (Gulino et al. 2024), Wayformer (Nayakanti et al. 2023), EasyChaffeur (Xiao et al. 2024), and PiDT (Zhou et al. 2025) we obtain consistently better ADE and overlap, and competitive offroad rates.

4.2 Evaluating the Classifier-Guided Actions

Having established that the DSS framework is useful, we now turn to evaluating the classifier-guided action selection proposed in Sec. 3.3. Here the agent only sees a single final (x, y) waypoint that marks its destination and must autonomously plan the intermediate trajectory to it. This represents a significantly more difficult and realistic AV problem setting.

Problem setting. A general conceptual challenge is how to incorporate a differentiable planning loss term encouraging the policy to reach the destination indicated by the last waypoint. That waypoint is fixed to the last physical timestep L which *may lie beyond the planning horizon* T. This creates a mismatch and prevents us from directly supervising the imagined location at time T with that waypoint beyond T. The difficulty arises from the difference between reaching the destination at a *particular* timestep vs reaching it at *any*

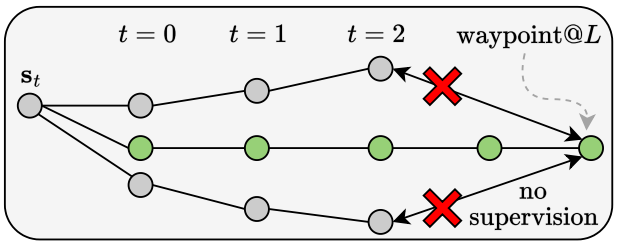

Figure 4: **Planning loss design**. The planning loss includes collision and offroad events. It does not supervise the last imagined location at time T with the last waypoint at time L.

timestep. To avoid this problem we do not supervise with the waypoint (see Fig. 4). The planning loss function is simply Eqn. 3. The agent is expected to have learned how to time its movement based on the scenarios seen during training.

Collisions and offroad events are sparse and occur only in some steps. By approximating them with a classifier, as described previously, we can minimize the likelihood of such events at test time. Table 5 shows the exact results. Crucially, *if the classifier is accurate, minimizing their probabilities during the imagination results in selecting those actions that avoid such events in the real physical trajectory*. As before, the search and the gradient updates are beneficial.

K	T	Grad	ADE ↓	overlap ↓	offroad ↓
1	20	✗	2.9872	0.1380	0.0781
1	20	✓	2.9817	0.1346	0.0737
4	10	✗	1.6016	0.0618	0.0641
4	10	✓	1.3861	0.0505	**0.0638**
4	20	✓	1.3235	0.0510	0.0782
8	10	✓	**1.2759**	**0.0442**	0.0762

Table 5: **Performance when planning only to minimize collisions and offroad.** Gradient step size is set to $\eta = (1000, 0.05)$. *Takeaway*: searching improves performance ($K = 1$ vs $K > 1$). A longer planning horizon T (rows 4–5) and gradients from the differentiable simulator (✓vs ✗), even though sparse, are still beneficial.

Importantly, even though the agent has not been explicitly trained to minimize overlap and collisions, the test time planning can nonetheless improve these metrics – ovelap rate improves by 9.37 perc. points (0.138 → 0.044), while ADE decreases by more than 2-fold (2.987 → 1.2759). Compared to state-of-the-art methods in Table 6, we attain significantly better ADE, which is more important than overlap and offroad alone, because it implicitly contains humanlike pacing, turning, and accelerating. EasyChaffeur and PiDT are directly trained to minimize collision and offroad and obtain better rates there. Unlike EasyChaffeur, our agent only sees the final (x, y) destination and has to decide where to go in the intermediate trajectory up to it. In this more realistic setting we obtain a 5-fold reduction in ADE compared to the other methods (1.27 vs PiDT's 6.99).

Model	ADE ↓	overlap ↓	offroad ↓
DecisionTransformer	8.3200	0.0362	0.0621
BC	3.6000	0.1120	0.1359
PiDT	6.9900	**0.0186**	0.0298
Wayformer	2.3800	0.1068	0.0789
EasyChaffeur-IL	—	0.0293	0.0280
EasyChaffeur-PPO	—	0.0443	**0.0216**
Ours (best configuration)	**1.2759**	0.0442	0.0762

Table 6: **Comparison with additional baselines**. *Takeaway*: our planning method outperforms state-of-the-art methods on ADE in a more realistic setting with less route conditioning.

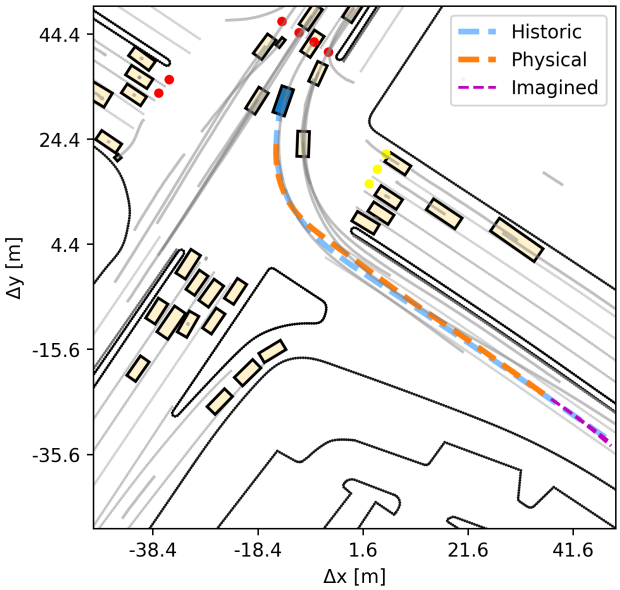

Figure 5: **Driving by planning**. The ego-vehicle is blue. All boxes are shown at their initial positions and the gray lines indicate their future motion (crossing lines do not imply collisions). Red dots are red lights. The ego-agent accurately navigates the intersection by periodically planning out its actions through imagination of the future (shown in purple).

Qualitative study. Fig. 5 shows a visualization of how the agent imagines the future at a particular point while driving in a crowded intersection. In general, we observe accurate trajectories that closely resemble the human ones. Further results and failure cases are discussed in the suppl. materials.

5 Conclusion

We have described DSS, a novel test-time planning framework for autonomous driving. In it, the ego-vehicle imagines how other agents will behave in the future and optimizes its own actions accordingly. It is implemented using differentiable simulation for the environment dynamics and learnable classifiers to approximate the non-differentiable collision and offroad computations. We have evaluated the framework in both tracking and autonomous path planning settings, showing strong gains compared to relevant methods. Our approach achieves better displacement rates, indicating more humanlike driving, and ensures that executed actions are selected using test-time optimization, rather than directly from a policy that is only expected to generalize. Potential future work includes transferring this idea to other simulators, additional dynamics models, and real-world situations.

Acknowledgements

This research was partially funded by the Ministry of Education and Science of Bulgaria (support for INSAIT, part of the Bulgarian National Roadmap for Research Infrastructure).

References

AlphaProof and AlphaGeometry teams. 2024. AI achieves silver-medal standard solving International Mathematical Olympiad problems. https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/. Published 25 July 2024.

Chen, L.; Lu, K.; Rajeswaran, A.; Lee, K.; Grover, A.; Laskin, M.; Abbeel, P.; Srinivas, A.; and Mordatch, I. 2021. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34: 15084–15097.

Dhariwal, P.; and Nichol, A. 2021. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34: 8780–8794.

Ettinger, S.; Cheng, S.; Caine, B.; Liu, C.; Zhao, H.; Pradhan, S.; Chai, Y.; Sapp, B.; Qi, C. R.; Zhou, Y.; et al. 2021. Large scale interactive motion forecasting for autonomous driving: The waymo open motion dataset. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 9710–9719.

Fawzi, A.; Balog, M.; Huang, A.; Hubert, T.; Romera-Paredes, B.; Barekatain, M.; Novikov, A.; R. Ruiz, F. J.; Schrittwieser, J.; Swirszcz, G.; et al. 2022. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930): 47–53.

Geilinger, M.; Hahn, D.; Zehnder, J.; Bächer, M.; Thomaszewski, B.; and Coros, S. 2020. Add: Analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Transactions on Graphics (TOG)*, 39(6): 1–15.

Gulino, C.; Fu, J.; Luo, W.; Tucker, G.; Bronstein, E.; Lu, Y.; Harb, J.; Pan, X.; Wang, Y.; Chen, X.; et al. 2024. Waymax: An accelerated, data-driven simulator for large-scale autonomous driving research. *Advances in Neural Information Processing Systems*, 36.

Hafner, D.; Lillicrap, T.; Ba, J.; and Norouzi, M. 2019. Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Heeg, J.; Song, Y.; and Scaramuzza, D. 2024. Learning Quadrotor Control From Visual Features Using Differentiable Simulation. *arXiv preprint arXiv:2410.15979*.

Holl, P.; Koltun, V.; and Thuerey, N. 2020. Learning to control pdes with differentiable physics. *arXiv preprint arXiv:2001.07457*.

Huang, Z.; Liu, H.; Wu, J.; and Lv, C. 2023. Differentiable integrated motion prediction and planning with learnable cost function for autonomous driving. *IEEE transactions on neural networks and learning systems*, 35(11): 15222–15236.

Karkus, P.; Ivanovic, B.; Mannor, S.; and Pavone, M. 2023. Diffstack: A differentiable and modular control stack for autonomous vehicles. In *Conference on robot learning*, 2170–2180. PMLR.

Li, S.; Huang, Z.; Chen, T.; Du, T.; Su, H.; Tenenbaum, J. B.; and Gan, C. 2023. Dexdeform: Dexterous deformable object manipulation with human demonstrations and differentiable physics. *arXiv preprint arXiv:2304.03223*.

Lin, X.; Huang, Z.; Li, Y.; Tenenbaum, J. B.; Held, D.; and Gan, C. 2022. Diffskill: Skill abstraction from differentiable physics for deformable object manipulations with tools. *arXiv preprint arXiv:2203.17275*.

Luo, L.; Liu, Y.; Liu, R.; Phatale, S.; Guo, M.; Lara, H.; Li, Y.; Shu, L.; Zhu, Y.; Meng, L.; et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Lutter, M.; Silberbauer, J.; Watson, J.; and Peters, J. 2021. Differentiable physics models for real-world offline model-based reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 4163–4170. IEEE.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533.

Moura, L. d.; and Ullrich, S. 2021. The Lean 4 theorem prover and programming language. In *Automated Deduction–CADE 28: 28th International Conference on Automated Deduction, Virtual Event, July 12–15, 2021, Proceedings 28*, 625–635. Springer.

Nachkov, A.; Paudel, D. P.; and Van Gool, L. 2025. Autonomous Vehicle Controllers From End-to-End Differentiable Simulation. In *2025 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Nachkov, A.; Paudel, D. P.; Zaech, J.-N.; Scaramuzza, D.; and Van Gool, L. 2026. Unlocking Efficient Vehicle Dynamics Modeling via Analytic World Models. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Nayakanti, N.; Al-Rfou, R.; Zhou, A.; Goel, K.; Refaat, K. S.; and Sapp, B. 2023. Wayformer: Motion forecasting via simple & efficient attention networks. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 2980–2987. IEEE.

Qiao, Y.-L.; Liang, J.; Koltun, V.; and Lin, M. C. 2020. Scalable differentiable physics for learning and control. *arXiv preprint arXiv:2007.02168*.

Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839): 604–609.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587): 484–489.

Snell, C.; Lee, J.; Xu, K.; and Kumar, A. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters, 2024. *URL https://arxiv.org/abs/2408.03314*, 11.

Song, Y.; Kim, S.; and Scaramuzza, D. 2024. Learning Quadruped Locomotion Using Differentiable Simulation. *arXiv preprint arXiv:2403.14864*.

Suo, S.; Regalado, S.; Casas, S.; and Urtasun, R. 2021. Trafficsim: Learning to simulate realistic multi-agent behaviors. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10400–10409.

Sutton, R. S. 1991. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4): 160–163.

Sutton, R. S.; Barto, A. G.; et al. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.

Toussaint, M. A.; Allen, K. R.; Smith, K. A.; and Tenenbaum, J. B. 2018. Differentiable physics and stable modes for tool-use and manipulation planning.

Trinh, T. H.; Wu, Y.; Le, Q. V.; He, H.; and Luong, T. 2024. Solving olympiad geometry without human demonstrations. *Nature*, 625(7995): 476–482.

Wiedemann, N.; Wüest, V.; Loquercio, A.; Müller, M.; Floreano, D.; and Scaramuzza, D. 2023. Training efficient controllers via analytic policy gradient. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, 1349–1356. IEEE.

Xiao, L.; Liu, J.-J.; Ye, X.; Yang, W.; and Wang, J. 2024. EasyChauffeur: A Baseline Advancing Simplicity and Efficiency on Waymax. *arXiv preprint arXiv:2408.16375*.

Xu, J.; Chen, T.; Zlokapa, L.; Foshey, M.; Matusik, W.; Sueda, S.; and Agrawal, P. 2021. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*.

Xu, J.; Kim, S.; Chen, T.; Garcia, A. R.; Agrawal, P.; Matusik, W.; and Sueda, S. 2023. Efficient tactile simulation with differentiability for robotic manipulation. In *Conference on Robot Learning*, 1488–1498. PMLR.

Zheng, L.; Son, S.; and Lin, M. C. 2022. Traffic-aware autonomous driving with differentiable traffic simulation. *arXiv preprint arXiv:2210.03772*.

Zhou, H.; Qin, Y.; Xu, D.; and Ji, Y. 2025. Physics-informed Imitative Reinforcement Learning for Real-world Driving. arXiv:2407.02508.