Differentiable Weights-Varying Nonlinear MPC via Gradient-based Policy Learning: An Autonomous Vehicle Guidance Example

Anonymous Author(s)

***Abstract*— Tuning Model Predictive Control (MPC) cost weights for multiple, competing objectives is labor-intensive. Derivative-free automated methods, such as Bayesian Optimization, reduce manual effort but remain slow, while Differentiable MPC (Diff-MPC) exploits solver sensitivities for faster gradient-based tuning. However, existing Diff-MPC approaches learn a single global weight set, which may be suboptimal as operating conditions change. Conversely, black-box Reinforcement Learning Weights-Varying MPC (RL-WMPC) requires long training times and a lot of data. In this work, we introduce gradient-based policy learning for Differentiable Weights-Varying MPC (Diff-WMPC). By backpropagating solver-in-the-loop sensitivities through a lightweight policy that maps look-ahead observations to MPC weights, our Diff-WMPC yields rapid, sample-efficient adaptation at runtime. Extensive simulation on a full-scale racecar model demonstrates that Diff-WMPC outperforms state-of-the-art static-weight baselines and is competitive with weights-varying algorithms, while reducing training time from over an hour to under two minutes relative to RL-WMPC. The learned policy transfers zero-shot to unseen conditions and, with quick online fine-tuning, reaches environment-specific performance. Project Website: `https://diffmpc.com`**

***Index Terms*— Model Predictive Control (MPC), Differentiable MPC, Weights-Varying MPC, Learning-based MPC, Differentiable Algorithms**

I. INTRODUCTION

Model Predictive Control (MPC) [1], [2] is a fundamental strategy for trajectory planning and control of complex robotic systems, optimizing a system's behavior over a future prediction horizon [3]. However, the closed-loop MPC performance critically depends on its problem parametrization, including the cost function weights [4], [5]. Manual weight tuning requires significant effort and expert knowledge. Existing automated weight-learning methods fall into two main families: derivative-free and differentiable gradient-based approaches. The former include Bayesian Optimization (BO) and Reinforcement Learning (RL), and were used to learn static and locally variable weights [6], [7]. However, they are typically limited by sample inefficiency [8], which hinders their use for rapid online adaptation. Conversely, recent advances in Differentiable MPC (Diff-MPC) have shown quicker gradient-based weights learning. However, the existing Diff-MPC examples [9], [10] learn a static set of weights, with no adaptation to local operating conditions, and address only linear or unconstrained nonlinear MPC.

As shown in Figure 1, we bridge this gap by introducing gradient-based policy learning to enable quick, sample-efficient online learning of cost weights within a constrained, nonlinear Differentiable Weights-Varying MPC (Diff-WMPC). By learning a policy that maps future observations to variable weight configurations, our approach allows the MPC to autonomously adapt its behavior to maximize task-level performance, even with zero-shot deployment in previously unseen scenarios.

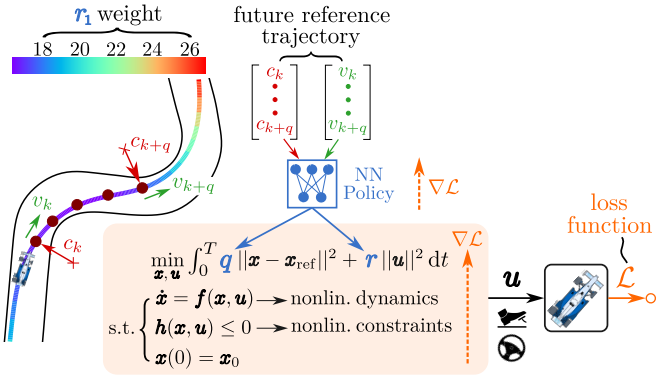

Fig. 1: Adaptive, context-aware, and sample-efficient cost weights optimization: A look-ahead neural policy network dynamically adapts and learns MPC cost weights online by backpropagating task-level performance gradients through a differentiable nonlinear MPC problem.

A. Related Work and Contributions

Different approaches have been proposed for MPC parameter learning. Our review focuses on derivative-free methods and on the emerging field of Diff-MPC.

Derivative-free methods treat the MPC problem as a black-box, and iteratively search for the best MPC parameters based on the observed performance. The authors of [11] performed particle swarm-based tuning of MPC parameters. In [6], BO was employed to adapt the MPC cost function and prediction model in different environmental conditions, while [12] used Multi-Objective BO (MOBO) to decrease the MPC model mismatch. Song et al. [13] proposed a policy-search to learn the MPC decision variables and parameters, for agile drone flight. In [4], constrained auto-tuning of MPC parameters was performed with Gaussian Process (GP) methods. The authors of [14] adapted the MPC constraints and dynamics by closed-loop iterative learning, in the context of autonomous racing [15]. RL was employed by [16], [17], [7] to learn static and locally variable cost function weights, and by [18], [19] to tune the MPC prediction horizon and meta-parameters. In [8], safe RL was conceived for weights-varying MPC, by constraining the RL actions (MPC weights) in a Pareto-optimal set obtained with MOBO.

However, the aforementioned derivative-free methods are typically data-hungry, requiring many interactions with the

environment and limiting their applicability for online training with real-world systems. To improve sample efficiency, recent works have explored Differentiable MPC, which leverages **gradient-based learning** of MPC parameters. In [9], analytical gradients of an MPC solution were exploited to learn the cost function and prediction model, while [20] devised an RL agent to learn the weights of a Diff-MPC for quadrotor control. However, [9], [20] used a linear MPC, with simple box control constraints and no state constraints. Recently, Frey et al. [21] showed how to compute the gradients of Nonlinear Model Predictive Control (NMPC) problems with respect to their parameters, by differentiating through the Karush-Kuhn-Tucker (KKT) optimality conditions with the ACADOS toolbox [22]. The authors of [23] proposed a software package building on ACADOS to compute policy gradients for RL agents, and showed an example of Q-learning of linear diff-MPC parameters. The only example of a Differentiable NMPC is [10], which solved a trajectory tracking problem with a nonlinear quadrotor model. However, their NMPC problem had no constraints, very few discretization points, and did not show local/ global online adaptation to new scenarios or dynamical models.

Critical Summary: To the best of our knowledge, the existing literature is limited by at least the following aspects:

- Derivative-free methods for weights-varying NMPC require extensive data and training time, limiting their practicality for online adaptation.
- Gradient-based weight learning with Differentiable MPC has been restricted to linear or unconstrained settings, and only for static weight sets.
- Prior differentiable MPC approaches have not demonstrated rapid online adaptation to previously unseen conditions or model mismatches.

To address these limitations, our main contributions are:

- We present the first Diff-WMPC framework for nonlinear systems with state and input constraints, enabling fast, context-dependent adaptation of MPC cost weights via a neural policy trained with solver gradients.
- We benchmark against Bayesian Optimization and weights-varying Reinforcement Learning-based MPC, achieving competitive or superior closed-loop performance with over an order-of-magnitude reduction in training time and sample usage.
- We demonstrate zero-shot transfer and rapid online adaptation in autonomous racing: a policy trained on one track and model is fine-tuned on a new, unseen track with higher-fidelity dynamics in two laps.

II. Methodology

Notation: We define a vector as $\mathbf{z} = [z_0, z_1, ..., z_n]^T \in \mathbb{R}^n$, containing n variables $z \in \mathbb{R}$. $\mathbf{Z} = [\mathbf{z}_0, \mathbf{z}_1, ..., \mathbf{z}_m] \in \mathbb{R}^{n \times m}$ defines a matrix composed of m concatenated vectors $\mathbf{z}$.

A. Nonlinear Model Predictive Control (NMPC) Problem

We formulate the general NMPC problem as a discrete-time finite-horizon Optimal Control Problem (OCP), with the state vector $\boldsymbol{x} \in \mathbb{R}^{n_x}$, initial state $\boldsymbol{x}_{init}$, control vector $\boldsymbol{u} \in \mathbb{R}^{n_u}$, stage cost function $l(\cdot)$, terminal cost function $m(\cdot)$, prediction horizon N, system dynamics $\boldsymbol{f}(\cdot)$, and nonlinear inequality and terminal constraints $\boldsymbol{h}(\cdot)$ and $\boldsymbol{h}^e(\cdot)$:

$$\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{u}} \quad & \int_{t=0}^{T_p} l\big(\boldsymbol{x}(t), \boldsymbol{u}(t); \boldsymbol{\theta}\big)\,\mathrm{d}t \;+\; m\big(\boldsymbol{x}(T_p); \boldsymbol{\theta}\big) \\ \text{subject to} \quad & \boldsymbol{x}(0) = \boldsymbol{x}_0, \\ & \dot{\boldsymbol{x}}(t) = \boldsymbol{f}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big), \quad t \in [0, T_p), \\ & \boldsymbol{h}\big(\boldsymbol{x}(t), \boldsymbol{u}(t)\big) \le \bar{\boldsymbol{h}}, \quad t \in [0, T_p), \\ & \boldsymbol{h}^e\big(\boldsymbol{x}(T_p)\big) \le \bar{\boldsymbol{h}}^e \end{aligned} \tag{1}$$

The MPC solution is $\mathbf{Z}^* = [\mathbf{X}^*, \mathbf{U}^*]$, with $\mathbf{X}^*$ and $\mathbf{U}^*$ being the computed states and controls for each timestep in the prediction horizon. At each time step, the first control action $\mathbf{u}_0^*$ is applied to the system, and the MPC is solved again from the new initial state.

B. Differentiating the NMPC Solution

In this work, we apply the recent theoretical advances in Differentiable Nonlinear MPC to compute the sensitivities of MPC solutions with respect to their cost function weights $\boldsymbol{\theta}$. We base our formulation on Frey et al. [21], using the ACADOS MPC toolbox [22]. The sensitivity calculation is the first step towards gradient-based weights learning with performance-related loss functions.

For a nonlinear constrained OCP, the solution set $\mathbf{Z}^*$ must satisfy the KKT optimality conditions [24], given by a nonlinear system of equalities and inequalities. Following [21], the sensitivities $\mathbf{S}$ of an OCP solution with respect to its weights can be obtained by applying the Implicit Function Theorem (IFT) to the KKT system, yielding:

$$\mathbf{S} = \frac{\partial \mathbf{w}^*}{\partial \boldsymbol{\theta}} = \frac{\partial (\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{s})^*}{\partial \boldsymbol{\theta}} = -\mathcal{M}_*^{-1} J_*, \tag{2}$$

where $\mathbf{S} \in \mathbb{R}^{n_w \times n_\theta}$, $\mathbf{w} = (\mathbf{z}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{s}) \in \mathbb{R}^{n_w}$ collects the states and controls ($\mathbf{z}$), Lagrange multipliers ($\boldsymbol{\lambda}, \boldsymbol{\mu}$), slack variables ($\mathbf{s}$), and $\mathbf{w}^*$ indicates a KKT solution. The sensitivity matrix $\mathbf{S}$ in (2) is the product of two Jacobians, $\mathcal{M}_*$ and J_*, which are derived from the solution of the KKT conditions with an interior-point method [21].

The application of the IFT to derive (2) requires standard regularity conditions at the solution, including the linear independence constraint qualification, the second-order sufficiency condition, and strict complementarity [21]. These conditions ensure that the solution is locally unique and stable with respect to changes in the MPC cost weights $\boldsymbol{\theta}$.

The sensitivity matrix $\mathbf{S}$ provides the relationship between a change in the cost weights $\boldsymbol{\theta}$ and the corresponding change in all optimal primal-dual variables $\mathbf{w}^*$, enabling the gradient-based learning approach presented in the following sections. Since the solution vector $\mathbf{w}^*$ contains the primal solution $\mathbf{z}^*$, which in turn is composed of the optimal state and control vectors, we can find the sensitivity of any specific part of the solution in the matrix $\mathbf{S}$.

Solving for the full sensitivity matrix in (2) is referred to as the *forward sensitivities* method. An alternative approach

provided by ACADOS is to solve for the *adjoint sensitivities*, which can be faster in learning tasks with a large number of parameters. In Section IV-G, we explain our reasoning for choosing the forward method.

C. Diff-MPC: Gradient-based Learning of the MPC Weights

We leverage the capability to extract the sensitivity of our optimal solution with regard to the weight vector $\frac{\partial \mathbf{z}^*}{\partial \boldsymbol{\theta}}$ from Equation (2) in the following sections to optimize our MPC cost weights via gradient descent.

This single gradient by itself does not define a learning framework, as it does not evaluate if a change in a specific way is beneficial for the high-level task objective of the MPC. To enable the intended learning capabilities, we have to connect this local gradient with a measure of the system performance, formulated as a loss function $\mathcal{L}$. As visualized in Figure 1, this function quantifies the closed-loop performance of the MPC on the given task, such as tracking a reference trajectory. By ensuring that the implementation of $\mathcal{L}$ is differentiable and depends on the optimal solution $\mathbf{z}^*$, the gradients of the loss function with regard to the optimal solution can be extracted as $\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \mathbf{z}^*}$. We utilize PyTorch to compute these gradients, although any framework with automatic differentiation can be implemented. Using the gradient chain rule and common intersection $\mathbf{z}^*$, we can combine the loss function gradient with the gradient we extract via ACADOS (Equation (2)) to receive the full end-to-end gradient connecting the high-level performance objective and parameter vector via the following equation:

$$\frac{\partial \boldsymbol{\mathcal{L}}}{\partial \boldsymbol{\theta}} = \frac{\partial \boldsymbol{\mathcal{L}}}{\partial \mathbf{z}^*} \frac{\partial \mathbf{z}^*}{\partial \boldsymbol{\theta}}. \tag{3}$$

This end-to-end gradient, combined with a gradient-based optimizer, enables the direct optimization of a set of MPC weights to minimize a loss function. We use the Adam optimizer [25], although various other methods, such as Standard Gradient Descent or RMSProp, are applicable.

Algorithm 1 presents our Diff-MPC learning framework. The process iterates over the total number of simulation steps N_{total} (line 3), representing the full training duration (e.g., multiple laps). At each timestep k, the algorithm solves the OCP with a set of cost weights $\boldsymbol{\theta}_k$ to compute the optimal solution $\mathbf{z}_k^*$ and the sensitivity matrix $\mathbf{S}_k$ (line 5). This solution is evaluated by a high-level loss function $\mathcal{L}_k$ (line 6), and the end-to-end gradient with respect to the weights, $\nabla_{\boldsymbol{\theta}} \mathcal{L}_k$, is derived (lines 7 - 8). From here, we configure the framework for two distinct learning strategies.

In the static-weights variant (Diff-MPC), we optimize a single parameter vector: $\boldsymbol{\theta}_k \equiv \boldsymbol{\theta}$ for all k (line 4). Here, we set the batch size N_{batch} to the length of one training episode (e.g., one lap). Consequently, the algorithm accumulates gradients (line 9) throughout the episode and applies a single gradient-based update (line 11) only when the episode is complete (line 10).

Algorithm 1 Differentiable (Weights-Varying) MPC

Require: Initial weight set $\boldsymbol{\theta}_0$; Reference trajectory $\boldsymbol{\Gamma}$; Number of simulation steps N_{total}; Batch size N_{batch}

1: Initialize $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}_0$ / $\boldsymbol{\pi}_{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta}_0$
2: $\mathbf{x}_k \leftarrow$ GetInitialState(); $\nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{accum}} \leftarrow \mathbf{0}$
3: **for** k to $(N_{\text{total}} - 1)$ **do** ▷ Iterate over all timesteps k
4: $\quad \boldsymbol{\theta}_k \leftarrow \begin{cases} \boldsymbol{\theta} & \text{// Static Weights} \\ \boldsymbol{\pi}_{\boldsymbol{\theta}}(\text{GetObservat}(\mathbf{x}_k)) & \text{// Weights Varying} \end{cases}$
5: $\quad (\mathbf{z}_k^*, \mathbf{S}_k) \leftarrow \text{SolveDiffMPC}(\mathbf{x}_k, \boldsymbol{\theta}_k)$
6: $\quad \mathcal{L}_k \leftarrow \text{ComputeLoss}(\mathbf{z}_k^*, \boldsymbol{\Gamma})$
7: $\quad \nabla_{\boldsymbol{\theta}} \mathcal{L}_k \leftarrow \text{WeightsGrad}(\mathcal{L}_k, \mathbf{z}_k^*, \boldsymbol{\theta}_k, \mathbf{S}_k)$
8: $\quad \nabla_{\boldsymbol{\theta}} \mathcal{L}_k \leftarrow \begin{cases} \nabla_{\boldsymbol{\theta}} \mathcal{L}_k & \text{// Static Weights} \\ \text{PolicyGrad}(\boldsymbol{\pi}_{\boldsymbol{\theta}}, \nabla_{\boldsymbol{\theta}} \mathcal{L}_k) & \text{// Varying} \end{cases}$
9: $\quad \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{accum}} \leftarrow \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{accum}} + \nabla_{\boldsymbol{\theta}} \mathcal{L}_k$
10: $\quad$ **if** $(k+1) \mod N_{\text{batch}} = 0$ **then**
$\quad$ ▷ Optimize weights/ policy every N_{batch} timesteps
11: $\quad\quad \boldsymbol{\theta}$ / $\boldsymbol{\pi}_{\boldsymbol{\theta}} \leftarrow \text{Optimizer}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{accum}})$
12: $\quad\quad \nabla_{\boldsymbol{\theta}} \mathcal{L}_{\text{accum}} \leftarrow 0$
13: $\quad$ **end if**
14: $\quad \mathbf{x}_{k+1} \leftarrow \text{SimulateStep}(\mathbf{x}_k, \mathbf{u}_k^*)$
15: **end for**
16: **return** $\boldsymbol{\theta}$ / $\boldsymbol{\pi}_{\boldsymbol{\theta}}$

D. Diff-WMPC: Gradient-based Policy Learning for Weights-Varying MPC

Utilizing a single set of MPC cost weights can limit the control performance, as different situations a system encounters during its deployment pose different challenges. To address these limitations, we extend Diff-MPC by incorporating a weights-varying policy network $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. Based on observations from the environment, the algorithm dynamically adapts good-performing cost weights $\boldsymbol{\theta}$ (line 4). Figure 1 visualizes our general methodology for this approach in a trajectory tracking scenario for autonomous vehicles.

Algorithm 1 presents the approach in detail, initializing the policy $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ in a manner that it outputs the initial MPC weights vector $\boldsymbol{\theta}_0$ for any input (line 1). We deploy a neural network to correlate observations and cost weights, though other options are possible. In each timestep k (line 3), the algorithm generates an observation $\mathbf{o}_k$ of the system's environment or state as input for the policy network. Based on this data, the network calculates the corresponding set of parameters $\boldsymbol{\theta}_k = \boldsymbol{\pi}_{\boldsymbol{\theta}}(\mathbf{o}_k)$ (line 4), used in the subsequent MPC solve step (line 5).

In comparison to the Diff-MPC, we additionally propagate the end-to-end gradient through the policy network (line 8), enabling the network to learn and improve its weight adaptation policy online. To avoid any momentary event having a disproportionate impact on the training performance, we accumulate the gradient over a user-specified number of timesteps. After reaching the specified batch size N_{batch} (line 10), the optimizer performs a gradient update step for the policy network (line 11). This mini-batching strategy allows us to perform frequent policy updates during one

training episode. All gradients are reset to zero (line 12), and the MPC continues operation using the improved policy network $\boldsymbol{\pi_\theta}$ in the next timestep.

III. Application Example: Trajectory Tracking with Autonomous Vehicles

To demonstrate the performance of our proposed methodology, we apply it to the domain of trajectory tracking in autonomous vehicles. Among autonomous vehicles, autonomous racing serves as a challenging application example, where the MPC needs to operate close to the vehicle's handling limits, dealing with combined longitudinal and lateral acceleration and nonlinear dynamics and constraints. Our experiments utilize a high-fidelity simulation environment built around the Dallara AV-24 racecar used in the Indy Autonomous Challenge competition [26]. Our vehicle dynamics model used inside the MPC is parametrized on telemetry data from the real car.

We adapt the NMPC Problem (1) for our application. The objective of the controller is to follow a pre-computed, time-optimal path and velocity reference trajectory, based on the implementation of [27]. The NMPC state vector is defined as $\mathbf{x} = [x_{\text{pos}}, y_{\text{pos}}, \psi, v_{\text{long}}, v_{\text{lat}}, \dot{\psi}, \delta_f, a]$, where x_{pos} and y_{pos} denote the global position, ψ the yaw angle, v_{long} and v_{lat} the longitudinal and lateral velocities, $\dot{\psi}$ the yaw rate, δ_f the front wheel steering angle, and a the longitudinal acceleration. The control input vector is $\mathbf{u} = [j, \omega_f]$, with j representing longitudinal jerk and ω_f the steering rate.

The prediction model $\boldsymbol{f}$ in (1) is a nonlinear single-track vehicle model with Pacejka Magic Formulas for tire forces based on [28] and extended to include lateral load transfer effects and aerodynamic drag and lift. The cost function is formulated as a nonlinear least-squares objective, with stage cost $l(\mathbf{x}, \mathbf{u}) = \frac{1}{2}\|\mathbf{y}(\mathbf{x}, \mathbf{u}) - \mathbf{y}_{\text{ref}}\|^2$ and terminal cost $m(\mathbf{x}) = \frac{1}{2}\|\mathbf{y}^e(\mathbf{x}) - \mathbf{y}^e_{\text{ref}}\|^2$. The weight vectors $\mathbf{q} = [q_{\text{n}}, q_\psi, q_{v_{\text{long}}}, q_{a_{\text{lat}}}]$ and $\mathbf{r} = [r_j, r_\omega]$ penalize deviations in lateral position n, yaw angle, longitudinal velocity, lateral acceleration a_{lat}, as well as control effort in jerk and steering rate, respectively. Reference vectors $\mathbf{y}_{\text{ref}} = [n_{\text{ref}}, \psi_{\text{ref}}, v_{\text{long}_{\text{ref}}}, a_{\text{lat}_{\text{ref}}}, 0, 0]$ and $\mathbf{y}^e_{\text{ref}} = [n_{\text{ref}}, \psi_{\text{ref}}, v_{\text{long}_{\text{ref}}}, a_{\text{lat}_{\text{ref}}}]$ specify desired trajectory and terminal states, with $a_{\text{lat}} = v_{\text{lon}}\dot{\psi}$. State and input constraints, including nonlinear combined longitudinal-lateral acceleration constraints, enforce the actuation limits and the track bounds. These constraints are derived from real-world racecar data [26] and based on the implementation in [28].

For our application, we connect the MPC to a high-level differentiable loss designed to minimize path and reference velocity deviations while imposing conditional penalties on the control actions:

$$\mathcal{L} = \alpha\|n - n_{\text{ref}}\|^2 + \beta\|v - v_{\text{ref}}\|^2 + \gamma P_j(j) + \delta P_\omega(\omega_f), \quad (4)$$

where $n - n_{\text{ref}}$ is the lateral and $v - v_{\text{ref}}$ is the velocity deviation from the reference trajectory. We define a general conditional penalty function for any variable x with threshold x_{th} and scaling parameter κ:

$$P_x(x) = \begin{cases} x^2, & \text{if } |x| \le x_{th} \\ \exp\left(\kappa(|x| - x_{\text{th}})\right) - 1, & \text{if } |x| > x_{\text{th}} \end{cases} \quad (5)$$

Specifically, we use $P_j(j)$ with $j_{\text{th}} = 8\,\frac{\text{m}}{\text{s}^3}$, $\kappa_j = 0.15$, and $P_\omega(\omega_f)$ with $\omega_{f,\text{th}} = 0.25\,\frac{\text{rad}}{\text{s}}$, $\kappa_\omega = 0.8$. This formulation applies a quadratic penalty within a safe region and transitions to a steeper exponential penalty for excessive control actions, promoting both smoothness and safety. The weights $\alpha, \beta, \gamma, \delta$ in (4) balance the contribution of each term. As all components are differentiable, gradients with respect to the MPC states and controls can be efficiently computed using automatic differentiation frameworks such as PyTorch. This enables end-to-end gradient computation via the chain rule (see Section II-C), directly linking the desired loss $\mathcal{L}$ to the static cost weights $\boldsymbol{\theta}$ (Diff-MPC) or weights-varying policy network $\boldsymbol{\pi}_\theta$ (Diff-WMPC) illustrated in Figure 2. As a result, we can optimize the controller's cost weights to maximize task-level performance.

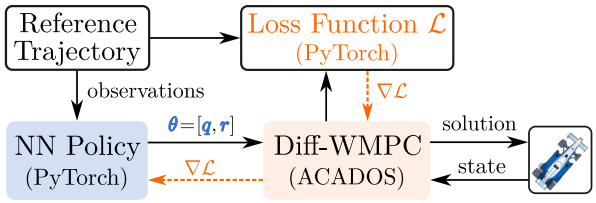

Fig. 2: Inside our simulation environment, the learning framework propagates the end-to-end loss function gradient through the MPC solver, to learn (and adapt) the MPC cost weights via a neural network policy.

In the stationary setting (static cost weights), based on the Diff-MPC formulation in Section II-C, we learn a single cost weight vector $\boldsymbol{\theta}$ for the entire track, accumulating gradients over an episode (one lap) before each update.

In contrast, our adaptive Diff-WMPC approach (Section II-D, Figure 2) employs a look-ahead policy network $\boldsymbol{\pi}_\theta$ that dynamically generates cost weights $\boldsymbol{\theta}_k$ at each timestep, conditioned on the upcoming reference trajectory. This enables context-aware adaptation, allowing the controller to anticipate and respond to varying reference features such as straights and corners. The policy network, presented in Figure 3, receives five future velocity values v_k and five curvatures c_k as input from the pre-computed reference trajectory, with a look-ahead horizon of 2.55 s. It processes these through two fully connected layers, L_1 and L_2, each containing 128 neurons and utilizing a softplus activation function. The output contains the six cost weights $\boldsymbol{\theta}_k = [\mathbf{q}_k, \mathbf{r}_k]$ for the MPC at each step. We backpropagate the end-to-end gradient through both the MPC and the policy network (Algorithm 1), enabling online local adaptation.

Remarks on Practical Stability and Feasibility: Locally adapting the cost weights during optimization may introduce additional dynamics into the NMPC problem. To mitigate potential instability, we employ specific architectural safeguards. We use a Softplus activation at the output of our

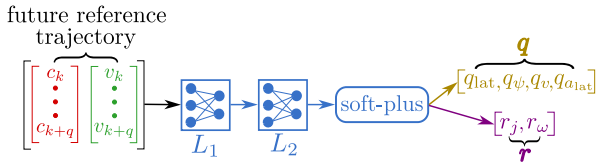

Fig. 3: Our weights-varying policy network uses observations from the environment (in our case, a future reference trajectory) to output the adapted MPC cost weights.

policy network to guarantee positive weights, thereby preserving a convex cost function. To avoid large weight updates that could destabilize the vehicle, we implement gradient clipping within the range ± 0.1, accumulate gradients over a batch of 10 timesteps, and apply a conservative learning rate of 2.9×10^{-5}. If the MPC solver fails to converge, a fallback mechanism reverts to the last feasible weights set. Structurally, we strictly decouple weights learning from the MPC constraints. A rigorous stability and recursive feasibility analysis under cost-weight adaptation remains for future work.

IV. Simulation Results: Diff-MPC & Diff-WMPC in Trajectory Tracking

A. Experimental Setup

To ensure a fair comparison, all algorithms share the same ACADOS-based MPC implementation in Python 3.9 and run on an AMD Ryzen 7950X CPU. The MPC uses the SQP_RTI solution method with the HPIPM quadratic programming solver, exact Hessian, and solves the KKT system to a tolerance of 10^{-6}. Table I summarizes the key parameters and initial cost weights. We select the learning rate, loss weights $\alpha, \beta, \gamma, \delta$ in (4), and batch size via Bayesian Optimization.

TABLE I: NMPC- and Learning Parameters

Parameter	Value
Simulator Discretization Time	0.02 s
MPC Discretization Time	0.075 s
MPC Prediction Horizon	2.55 s
Sensitivity Shooting Node Index	1
Initial $\mathbf{q}$ Weights	[2.5, 2.9, 2.0, 5.0]
Initial $\mathbf{r}$ Weights	[4.3, 6.8]
$\alpha, \beta, \gamma, \delta$	$2.5 \times 10^{-5}, 2.25$ $2.5 \times 10^{-7}, 9 \times 10^{-3}$
Diff-WMPC learning rate lr	2.9×10^{-5}
Diff-WMPC batch size B	10

We evaluate performance under both nominal and mismatched dynamics using two simulation models:

Simulator 1 (Simplified Dynamics): A nonlinear single-track model with a simplified Magic-Formula lateral tire law [28]. This is used both as MPC prediction model and as a simulation model to be driven, in our nominal experiments.

Simulator 2 (Higher-Fidelity Dynamics): In our adaptation experiments, we let our MPC control a more complex single-track vehicle simulator, with Pacejka '96 (MF96) lateral forces and combined-slip scaling. It models per-wheel slip angles and loads, captures load transfer via roll-center geometry, includes velocity-dependent downforce and drag, distributes traction/braking across axles, and couples longitudinal–lateral forces. Simulator 2 provides a higher fidelity under high-acceleration maneuvers, while Simulator 1 is lighter and well-suited for efficient training.

B. Benchmark Algorithms

1) Human Expert manually parametrized MPC (HE-MPC): This baseline approach reflects the traditional practice of manually tuning a single set of MPC cost weights by an experienced human expert.

2) Multi-Objective Bayesian Optimization MPC (MOBO-MPC): We use constrained Multi-Objective Bayesian Optimization MPC (MOBO-MPC) as a benchmark to optimize a single set of NMPC cost weights. Independent GPs model each objective, a GP classifier with Dirichlet likelihood enforces feasibility, and sampling uses Expected Hypervolume Improvement. In this work, we adopt the same framework as [8] but modify the NMPC formulation and dynamics from a passenger vehicle to a racecar at the limits, optimize a single global weight vector for the full track, and expand to four objectives: maximum lateral deviation error, RMS speed error, standard deviation of steering rate, and standard deviation of longitudinal jerk, while maintaining MPC-based feasibility enforcement.

3) Deep Reinforcement Learning Weights-Varying MPC (RL-WMPC): We adopt the RL-WMPC framework from [16] as benchmark algorithms for online adaptation of NMPC cost weights. In the RL-WMPC of [16], a continuous-action Deep RL agent updates weights at predefined switching intervals using rewards based on multiple control objectives, observing buffered MPC signals such as lateral deviation, velocity error, and jerk. In this work, we deploy this approach using continuous actions, racecar dynamics at the handling limits, and look-ahead trajectories. The RL-WMPC uses the identical observation inputs as our Diff-WMPC.

C. Diff-WMPC: Learning Control Policies in under Two Minutes

We benchmark and train all algorithms on the Monza racetrack using a minimum-lap-time reference trajectory, exploiting the full combined tire friction envelope and reaching peak velocities of $\approx 79\ \frac{\text{m}}{\text{s}}$.

TABLE II: Training time comparison: Our Diff-WMPC achieves **over an order-of-magnitude speedup** in both wall-clock time and sample usage compared to the RL-WMPC.

Algorithm	Training Time [s]	Samples
MOBO-MPC	1071	460,278
Diff-MPC (Ours)	**235**	**88,996**
RL-WMPC	3885	1,000,000
Diff-WMPC (Ours)	**101**	**36,778**

Table II reports wall-clock training time and the number of forward simulation samples required to obtain the final tuned cost weights/policies. Our gradient-based methods yield substantial efficiency gains: (i) for static cost weights, Diff-MPC reduces optimization time by 78.1 % (4.6× faster)

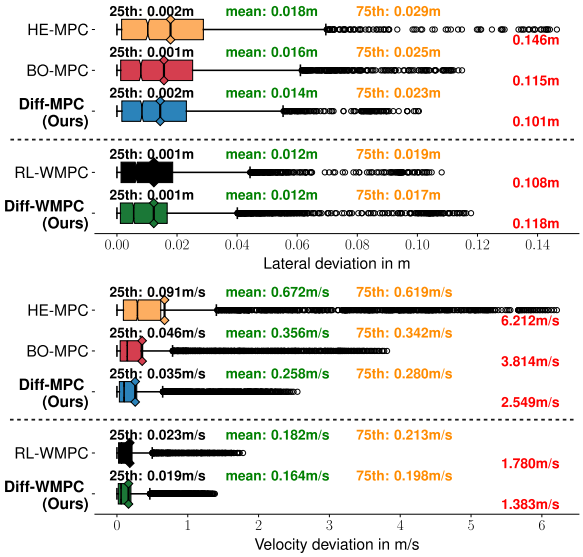

Fig. 4: One lap on the Monza racetrack in Simulator 1, in a setting with no model mismatch. Dashed lines separate static-weights and weights-varying approaches. From scratch, our **Diff-WMPC converges within 6 training laps.**

and uses 5.2× fewer samples than MOBO-MPC; (ii) for weight-varying policies, Diff-WMPC trains **38.5× faster than RL-WMPC (97.4 % reduction)** with 27.2× fewer samples, converging in under two minutes, equivalent to 6 laps or 630 s of real driving. Thus, Diff-WMPC achieves over an order-of-magnitude speedup in wall-clock time and sample usage compared to the RL-WMPC algorithm.

Moreover, our Diff-MPC and Diff-WMPC attain closed-loop performance that matches or surpasses state-of-the-art baselines. Figure 4 compares the lateral and velocity tracking errors of our differentiable methods with the benchmarks: (i) static global weights optimization and (ii) weights-varying approaches (separated by dashed lines). Here, experiments are performed in a perfect-model setting (Simulator 1), where the simulator dynamics match the MPC internal prediction model. Our differentiable controllers consistently outperform the manually tuned baseline, and are competitive with or superior to the BO and RL-based counterparts. This comparison of performance and training time indicates that solver-in-the-loop gradient information significantly accelerates convergence without compromising control quality. Additionally, these experiments clearly demonstrate the performance improvements that weights-varying, particularly differentiable weights-varying, approaches enable compared to their static-weight counterparts.

D. Interpreting the Learned Weight Adaptation Policy

Figure 5 illustrates two representative MPC weights our weight-varying policy dynamically adapts along one lap in Monza. The velocity-error weight increases on straights for accurate speed tracking, and decreases in corners to accommodate lateral limits. Conversely, the steering-rate weight relaxes on straights and tightens through turn-in and

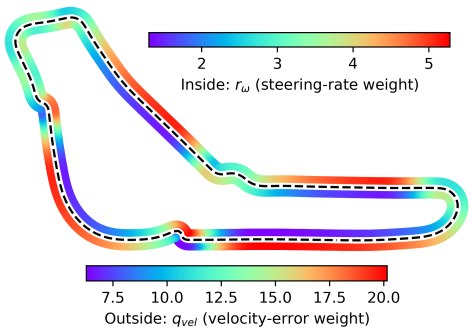

Fig. 5: Proactive adaptation of MPC cost weights over one Monza lap: Heatmaps of two representative weights show how the look-ahead policy anticipates upcoming track features and driving phases, smoothly adjusting cost weights before straights, corners, braking, and acceleration phases. The vehicle travels clockwise.

apex to suppress aggressive inputs and jerk. Our look-ahead policy adjusts each weight before entering the main corners, indicating anticipatory adaptation.

E. Training with Model Mismatch

In real-world applications, the internal prediction model of the MPC inevitably deviates from the true system dynamics. To test how the controllers generalize to such a mismatch, we train them in a setting where the controller's internal prediction model (based on Simulator 1) is no longer identical to the environment's true dynamics (Simulator 2), forcing the algorithms to cope with the discrepancy. Figure 6 shows that both Diff-MPC and Diff-WMPC generalize reliably and achieve competitive or better performance than the bench-

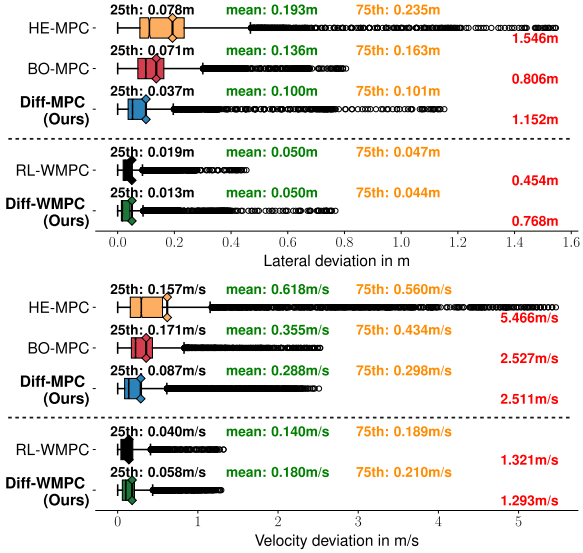

Fig. 6: Performance at Monza under a model mismatch, where the controller's internal prediction model (Simulator 1) differs from the high-fidelity Simulator 2 to be driven. Our methods show robust behavior and remain competitive.

mark algorithms. The improvements of our weights-varying policy over static cost weights are even more visible, as the Diff-WMPC **reduces the mean lateral deviation compared to the best static weight method by 50 %** and the mean velocity error by 37.5 %.

F. Zero-Shot Generalization and Quick Adaptation to unseen Conditions

We now deploy our Diff-WMPC policy trained on Monza with Simulator 1 directly on the unseen Laguna Seca circuit under the higher-fidelity Simulator 2 dynamics, inducing simultaneous model mismatch and a new track layout. Figure 7 shows the results: with only two laps of online fine-tuning (27 s in simulation and ≈178 s equivalent real-world driving), our policy adapts to Laguna Seca and matches the performance of a controller trained from scratch on this environment, demonstrating fast, data-efficient transfer.

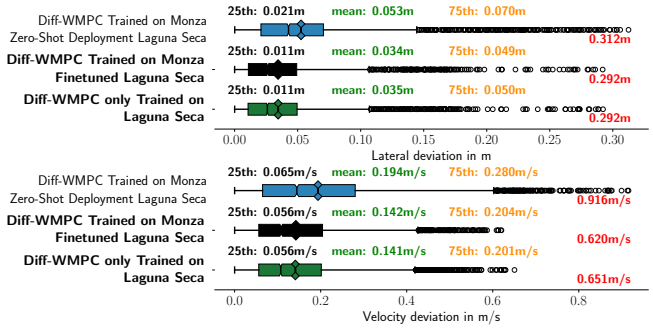

Fig. 7: Zero-Shot to Rapid Adaptation: the learned Diff-WMPC policy successfully transfers from Monza in Simulator 1 to the unseen Laguna Seca track in Simulator 2, and **adapts within 27 s** (≈178 s real driving) to reach environment-specific performance.

G. Sensitivity Analysis, Ablation Study, and Compute Time

a) Robustness to initialization: We first assess robustness to the initialization of the MPC cost weights at Monza under model mismatch. Across 30 random initializations (six weights sampled uniformly in the range $[0, 10]$), pre-training performance varies widely. Yet all runs converge to a near-identical post-training solution (Table III), indicating low sensitivity and strong robustness to the initialization choice.

TABLE III: Robustness to initialization: 30 random MPC cost-weight sets all converge to a high-performance solution.

Parameter / Metric	**Value**
Number of Samples	30
Initial MPC Weight Range	$6 \times [0, 10]$
Pre-Training RMSE Range (Lateral Deviation) [m]	$0.152 - 0.422$
Post-Training RMSE Range (Lateral Deviation) [m]	$0.090 - 0.091$
Pre-Training RMSE Range (Velocity Deviation) [m/s]	$0.609 - 1.561$
Post-Training RMSE Range (Velocity Deviation) [m/s]	$0.345 - 0.369$
Pre-Training Mean Accumulated Loss (per Lap)	886.8 ± 503.4
Post-Training Mean Accumulated Loss (per Lap)	90.1 ± 0.46

b) Choice of sensitivity node: Using Equation (2), we compute the MPC sensitivities with respect to the first shooting node of the prediction horizon. Training with nodes n.1, 10, and 20 yields post-training lateral RMSEs of $0.090\,\mathrm{m}$, $0.108\,\mathrm{m}$, and $0.191\,\mathrm{m}$, respectively (velocity RMSEs $0.351\,\frac{\mathrm{m}}{\mathrm{s}}$, $0.504\,\frac{\mathrm{m}}{\mathrm{s}}$, and $0.766\,\frac{\mathrm{m}}{\mathrm{s}}$). Hence, node 1 strikes the best balance between gradient informativeness .

c) Ablation study: The first two box-plots in Figure 8 remove either the future velocity v_{ref} or the future curvature c_{ref} from our Diff-WMPC policy's look-ahead context (Figure 3), showing performance degradation and confirming the relevance of both signals. Likewise, restricting learning to only state or only control weights (third and fourth box-plots) underperforms learning the full weight vector.

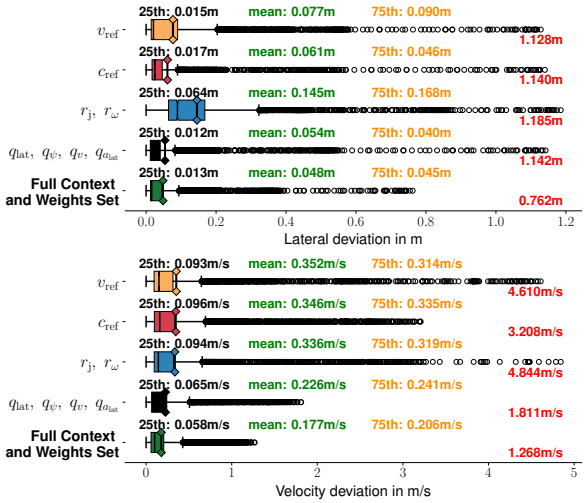

Fig. 8: Ablation study: removing policy context (future velocity v_{ref} or future curvature c_{ref}) or restricting adaptation to only state or control weights degrades performance.

d) Computational efficiency: We compare the mean runtime during training and inference (Figure 9). Our computational efficiency allows us to employ the forward sensitivity method, providing the full sensitivity matrix for more flexible analysis and experimentation. Online training with

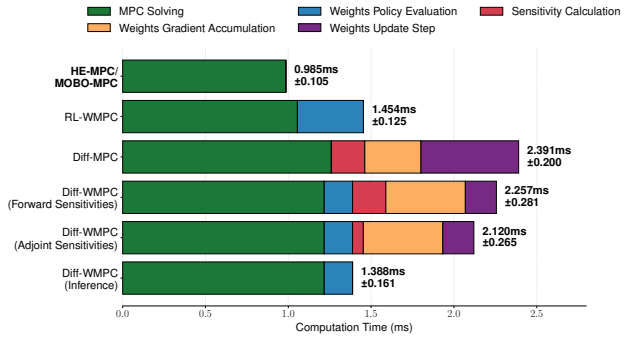

Fig. 9: A comparison of the mean computation times shows the additional resources the online training with Diff-MPC and Diff-WMPC requires. After training, we deactivate the learning-related tasks, reducing the runtime significantly.

Diff-MPC/ Diff-WMPC adds overhead from learning-related tasks. After training, these components are disabled and only the MPC and a lightweight policy remain. At inference-time Diff-WMPC achieves a very quick sub-$1.4\,\mathrm{ms}$ latency.

V. CONCLUSIONS

This work addresses the automatic optimization of cost weights for high-performance nonlinear MPC under changing operating conditions. We present Differentiable Weights-Varying MPC (Diff-WMPC), the first framework to combine solver-in-the-loop gradient learning with dynamic, context-dependent weight adaptation for systems with nonlinear dynamics and constraints. Our approach builds systematically from static-weight Diff-MPC to adaptive Diff-WMPC, leveraging a neural policy to adjust MPC cost weights online.

Extensive experiments in our autonomous racing simulation environment show that our methods match or outperform state-of-the-art benchmarks, including BO and RL-WMPC. All while delivering a paradigm shift in efficiency, where Diff-MPC reduces the training time by $78\,\%$ compared to MOBO-MPC, and Diff-WMPC achieves a $97.4\,\%$ reduction and requires 27 times fewer samples than RL-WMPC. We furthermore demonstrate how our adaptive weights-varying policy significantly improves trajectory tracking performance over the best static-weight approach, reducing mean lateral deviation by up to $50\,\%$ and mean velocity error by $37.5\,\%$.

While our approach enables rapid and robust adaptation, several limitations remain. Gradient-based optimization may get stuck in local minima, and global optimality is not guaranteed. Online weight adaptation in constrained NMPC depends on differentiability assumptions, and active set changes can introduce non-differentiable kinks, causing noisy gradients and possible instability or infeasibility. Rapid weight updates may also reduce stabilizing terms or push trajectories toward constraint boundaries. We can mitigate these risks by projecting weights into safe sets, using small learning rates, regularizing updates, and monitoring solver conditioning. Future work should address formal safety guarantees , extend learning to other MPC components, and demonstrate the methodology on a physical platform.

REFERENCES

[1] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. Volume 3, 2020, pp. 269–296, 2020.

[2] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat, "Mpc-based approach to active steering for autonomous vehicle systems," *International journal of vehicle autonomous systems*, 2005.

[3] F. Nan, S. Sun, P. Foehn, and D. Scaramuzza, "Nonlinear mpc for quadrotor fault-tolerant control," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5047–5054, 2022.

[4] A. G. Puigjaner, M. Prajapat, A. Carron, A. Krause, and M. N. Zeilinger, "Performance-driven constrained optimal auto-tuner for mpc," *IEEE Robotics and Automation Letters*, 2025.

[5] D. Kostadinov and D. Scaramuzza, "Online weight-adaptive nonlinear model predictive control," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 1180–1185.

[6] L. P. Fröhlich, C. Küttel, E. Arcari, L. Hewing, M. N. Zeilinger, and A. Carron, "Contextual tuning of model predictive control for autonomous racing," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.

[7] K. Lee, D. Isele, E. A. Theodorou, and S. Bae, "Spatiotemporal costmap inference for mpc via deep inverse reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, 2022.

[8] B. Zarrouki, M. Spanakakis, and J. Betz, "A safe reinforcement learning driven weights-varying model predictive control for autonomous vehicle motion control," in *2024 IEEE Intelligent Vehicles Symposium (IV)*, 2024, pp. 1401–1408.

[9] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Advances in neural information processing systems*, vol. 31. Curran Associates, Inc., 2018.

[10] R. Tao, S. Cheng, X. Wang, S. Wang, and N. Hovakimyan, "DiffTune-MPC: Closed-Loop Learning for Model Predictive Control," *IEEE Robotics and Automation Letters*, vol. 9, no. 8, Aug. 2024.

[11] A. Kapnopoulos and A. Alexandridis, "A cooperative particle swarm optimization approach for tuning an mpc-based quadrotor trajectory tracking scheme," *Aerospace Science and Technology*, vol. 127, p. 107725, 2022.

[12] G. Makrygiorgos, A. D. Bonzanini, V. Miller, and A. Mesbah, "Performance-oriented model learning for control via multi-objective bayesian optimization," *Computers & Chemical Engineering*, vol. 162, 2022.

[13] Y. Song and D. Scaramuzza, "Policy search for model predictive control with application to agile drone flight," *IEEE Transactions on Robotics*, vol. 38, no. 4, 2022.

[14] M. Piccinini, S. Taddei, J. Betz, and F. Biral, "Kineto-dynamical planning and accurate execution of minimum-time maneuvers on three-dimensional circuits," in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, 2025.

[15] J. Betz, H. Zheng, A. Liniger, U. Rosolia, P. Karle *et al.*, "Autonomous vehicles on the edge: A survey on autonomous vehicle racing," *IEEE Open Journal of Intelligent Transportation Systems*, vol. 3, 2022.

[16] B. Zarrouki, V. Klös, N. Heppner, S. Schwan, R. Ritschel, and R. Voßwinkel, "Weights-varying mpc for autonomous vehicle guidance: a deep reinforcement learning approach," in *2021 European Control Conference (ECC)*, 2021.

[17] M. Mehndiratta, E. Camci, and E. Kayacan, "Automated tuning of nonlinear model predictive controller by reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[18] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Reinforcement learning of the prediction horizon in model predictive control," *IFAC-PapersOnLine*, vol. 54, no. 6, pp. 314–320, 2021, 7th IFAC Conference on Nonlinear Model Predictive Control NMPC 2021.

[19] E. Bøhn, S. Gros, S. Moe, and T. A. Johansen, "Optimization of the model predictive control meta-parameters through reinforcement learning," *Engineering Applications of Artificial Intelligence*, vol. 123, p. 106211, 2023.

[20] A. Romero, Y. Song, and D. Scaramuzza, "Actor-Critic Model Predictive Control," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*. Yokohama, Japan: IEEE, May 2024.

[21] J. Frey, K. Baumgärtner, G. Frison, D. Reinhardt, J. Hoffmann *et al.*, "Differentiable Nonlinear Model Predictive Control," 2025.

[22] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. V. Duijkeren *et al.*, "acados—a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, vol. 14, no. 1, Mar. 2022.

[23] D. Reinhardt, K. Baumgärtner, J. Frey, M. Diehl, and S. Gros, "Mpc4rl - a software package for reinforcement learning based on model predictive control," in *2024 IEEE 63rd Conference on Decision and Control (CDC)*, 2024.

[24] W. Karush, *Minima of Functions of Several Variables with Inequalities as Side Conditions*. Basel: Springer Basel, 2014.

[25] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, 2014.

[26] J. Betz, T. Betz, F. Fent, M. Geisslinger, A. Heilmeier *et al.*, "Tum autonomous motorsport: An autonomous racing software for the indy autonomous challenge," *Journal of Field Robotics*, vol. 40, no. 4, 2023.

[27] M. Rowold, L. Ögretmen, U. Kasolowsky, and B. Lohmann, "Online time-optimal trajectory planning on three-dimensional race tracks," in *2023 IEEE Intelligent Vehicles Symposium (IV)*, 2023, pp. 1–8.

[28] B. Zarrouki, C. Wang, and J. Betz, "A stochastic nonlinear model predictive control with an uncertainty propagation horizon for autonomous vehicle motion control," in *2024 American Control Conference (ACC)*, 2024.