Real-Time Deep-Learning-Driven Parallel MPC

Roman Kohút, Erika Pavlovičová, Kristína Fedorová, Juraj Oravec, Michal Kvasnica

***Abstract*— A novel real-time approximated MPC control policy based on deep learning is proposed to address the high computational burden of model predictive control (MPC) for large-scale systems and those with fast dynamics. This control method approximates the optimal solution of the distributed optimization problems in the ALADIN-based parallel MPC design framework, resulting in a highly effective approach that outperforms other well-known methods for solving the MPC design problem. The numerical case study shows promising results, demonstrating the potential of this approach for real-time implementation.**

I. Introduction

Model predictive control (MPC), since its origin in the 1970s, has become one of the most popular methods for advanced process control in many industrial applications, see [1] and references therein. Besides many advantages, including optimal control of multiple-input multiple-outputs (MIMO) systems concerning technological constraints, MPC brought challenges for researchers worldwide. A relevant challenge is using MPC for real-time control of complex industrial systems. As the computation of optimal control is performed iteratively in each time step, the industrial implementation can be limited by solving the optimization problem for a given complex system. A perspective approach to reduce the time necessary for the real-time evaluation of optimal control action is presented by explicit MPC [2]. However, this approach is limited, as the number of the critical regions over which the piece-wise affine solution map of the parametric quadratic program is defined grows exponentially with the number of constraints [3]. Another perspective control approach arises with [4] initiating the implementation of distributed optimization. The introduction of distributed optimization in the framework of MPC enables formulating the large-scale MPC problem into a set of smaller subproblems, e.g., see [5]. Consequently, the distributed control policy significantly reduces the overall computational burden. These subproblems are efficiently solved in parallel by various off-the-shelf solvers, e.g., see [6] and references therein. Tailored solvers focused on such distributed optimization include `OSQP` [7] or `ALADIN-`α [8], to name a few.

Authors are with the Institute of Information Engineering, Automation, and Mathematics, Faculty of Chemical and Food Technology, Slovak University of Technology in Bratislava, Radlinského 9, SK-812 37, Bratislava, e-mail: `roman.kohut@stuba.sk`.

This research is funded by the European Commission under the grant no. 101079342 (Fostering Opportunities Towards Slovak Excellence in Advanced Control for Smart Industries). The authors gratefully acknowledge the contribution of the Scientific Grant Agency of the Slovak Republic under the grants 1/0490/23, 1/0297/22, and the Slovak Research and Development Agency under the project APVV-20-0261.

Recently, some of the most promising methods of distributed optimization include widely-used the alternating direction method of multipliers (ADMM) [4] and the augmented Lagrangian-based alternating direction inexact Newton method (ALADIN) [9]. These methods have been utilized to solve the real-time control problem using MPC, see [10] or [11]. The distributed optimization was successfully implemented into the framework of the parallel explicit MPC in [12], [13], the robust/tube parallel MPC in [14], and addressing the state constraints in [15]. Even though these control methods can significantly decrease the runtimes and/or memory consumption, the requirement to implement the distributed MPC controllers on embedded hardware keeps this research in high interest. The perspective approach considers the application of neural networks (NNs) with offline training to approximate the optimal control law of MPC, see [16], [17]. Alternatively, reinforcement learning can be employed to achieve data-driven nonlinear optimal control without the need for a model of the system [18]. Another widely-used concept is utilizing NNs to warm start MPC optimizer to improve the convergence [19], or incorporating the NNs to approximate the parts of the distributed optimization algorithms, see [20], [21], and references therein.

The main contribution of this paper is to introduce a novel real-time control policy based on deep-learning approximated solutions of the distributed optimization problems into the ALADIN-based parallel MPC design framework. The NNs are tuned to fit the distributed MPC control law close to optimality. Compared to explicit MPC, the memory footprint of the NNs is negligible compared to the corresponding explicit solution maps. On the other hand, in the case of large-scale systems, the formulation of the implicit (non-explicit) MPC design problem increases memory demands compared to the associated approximation based on the NNs. Simultaneously, the real-time evaluation of NNs is competitive.

II. Preliminaries

The subsequent sections briefly outline the design process for Model Predictive Control (MPC) and the parallel MPC, including their advantages and limitations. These sections also motivate the fundamental principles of a proposed novel approach addressing the implementation challenges of the non-distributed MPC.

A. Nominal MPC design problem

The receding horizon MPC policy is a control method that repeatedly solves optimization problems in each sampling period. Its primary goal is determining the optimal control

law within predefined constraints over the prediction horizon. This approach considers the MPC formulated as a problem of the quadratic programming (QP) with a linear time-invariant (LTI) model and linear constraints

$$\begin{aligned} J = \min_{u,x} \sum_{k=0}^{N-1} \left(\|x(k+1)\|_Q^2 + \|u(k)\|_R^2 \right) + \|x(N)\|_P^2, \\ \text{s.t. } \forall k \in \{0, \dots, N-1\}, \\ x(k+1) = Ax(k) + Bu(k), \\ x(k+1) \in \mathbb{X}, \quad u(k) \in \mathbb{U}, \\ x(0) = x_t, \end{aligned} \tag{1}$$

where N is the length of a prediction, $Q \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ is a symmetric positive-definite weight matrix of the system states $x(k) \in \mathbb{R}^{n_\mathrm{x}}$, $R \in \mathbb{R}^{n_\mathrm{u} \times n_\mathrm{u}}$ is a symmetric positive-definite weight matrix of the control inputs $u(k) \in \mathbb{R}^{n_\mathrm{u}}$, $P \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ is a symmetric positive-definite weight matrix of a terminal penalty $x(N) \in \mathbb{R}^{n_\mathrm{x}}$, and it is obtained as solution of the matrix Riccati equation. Matrix $A \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{x}}$ is a system matrix and $B \in \mathbb{R}^{n_\mathrm{x} \times n_\mathrm{u}}$ is an input matrix. This paper assumes that the pair (A, B) is asymptotically stabilizable for the solution to the matrix Riccati equation to exist [5]. One of the significant advantages of the MPC is its ability to consider the technological constraints represented as set $\mathbb{X}$ for state variables and $\mathbb{U}$ for control inputs, respectively. Sets $\mathbb{X}$, $\mathbb{U}$ include origin in their strict interiors.

At every time step, the current measurement of system state variables $x_t \in \mathbb{R}^{n_\mathrm{x}}$ is used to determine the optimal values of control inputs and state variables. Solving the MPC problem for a large-scale system with the length prediction horizon can be a time-consuming task, and it might be impossible to find a solution within a sampling time T_s). To considerably decreases the runtime needed to achieve the optimal solution, distributed optimization is a viable option, particularly for block-structured optimization problems like the MPC problem.

B. ALADIN-based parallel MPC design

ALADIN is a general method for solving distributed optimization problems that generalize the augmented Lagrangian and sequential quadratic programming (SQP) methods, as proposed in [9]. In [15], the nominal MPC is distributed into separate control steps within the prediction horizon. Then, the cost function J in (1) is split into individual stage cost functions $\ell = f(x(k), u(k))$, evaluating variable pairs $x(0)$ and $u(0)$, as well as a separate stage cost for the terminal control step determined by $x(N)$. This approach increases the number of solving $(N+1)$ agents and leads to three distinct optimization problem structures.

By introducing a simple shift in the considered decision variables of the stage cost functions $\ell := f(x(k+1), u(k))$, it is possible lower the number of decoupled subproblems to N solving agents. Moreover, such a distribution exhibits symmetric structured properties of the cost function J_D across the entire prediction horizon, see Figure 1.

The MPC formulation in (1) is divided into particular time instants k of the prediction horizon and the i-th block-structured system states, resulting in a distribution of the original problem in both time and space domains. As a result, each agent $n_{i,k}$ in the distributed MPC problem solves a corresponding decoupled optimization problem. In general, these subproblems are optimized in parallel. Therefore, the decoupled QPs are formulated as augmented Lagrangian functions in the form

$$\begin{aligned} J_\mathrm{D} = \min_{u_i, x_i} \; & \|x_i(k+1)\|_{Q_i}^2 + \|u_i(k)\|_{R_i}^2 \\ & + \lambda_i(k)^\top (x_i(k+1) - z_i(k+1)) \\ & + \frac{\rho}{2} \|x_i(k+1) - z_i(k+1)\|_Q^2 \\ & + \frac{\rho}{2} \|u_i(k) - v_i(k)\|_R^2, \\ \text{s.t. } & x_i(k+1) = A_i x(k) + B_i u(k), \\ & x_i(k+1) \in \mathbb{X}, \quad u_i(k) \in \mathbb{U}, \\ & x(k) = z(k), \\ & u_j(k) = v_j(k) \quad , \forall j \neq i, \end{aligned} \tag{2}$$

where the decision variables are $x_i(k+1) \in \mathbb{R}^{n_{x_i}}$ and $u_i(k) \in \mathbb{R}^{n_{u_i}}$ represents state and control inputs for define step $k = 0, \dots N-1$ and state space distribution $j = i = 1, \dots, S$. The weight matrix $Q_i \in \mathbb{R}^{n_{x_i} \times n_{x_i}}$, $R_i \in \mathbb{R}^{n_{u_i} \times n_{u_i}}$ and model matrix $A_i \in \mathbb{R}^{n_{x_i} \times n_x}$, $B_i \in \mathbb{R}^{n_{u_i} \times n_u}$ are partial matrices of original Q, R, and A, B respectively. The $\lambda_i \in \mathbb{R}^{n_{x_i}}$ is a Lagrangian multiplayer, also referred to as a dual variable, and $\rho > 0$ is a penalty parameter. The last step of the prediction horizon $x_i(k+1) = x_i(N)$ represents the only variation in the structure of parallel MPC. Including a terminal set of constraints $\mathbb{X}_\mathrm{P}$ with the terminal penalty P_i found by solving a matrix algebraic Riccati equation [22].

Each decoupled QP in (2) is then solved with initial guesses of $\lambda_i = [\lambda_i^\top(0), \lambda_i^\top(1), \dots, \lambda_i^\top(N)]^\top$ dual variable, as well as $z_i = [z_i^\top(0), z_i^\top(1), \dots, z_i^\top(N)]^\top$ and $v_i = [v_i^\top(0), v_i^\top(1), \dots, v_i^\top(N)]^\top$ representing state and control trajectories. From the initial guess, the algorithm consecutively solves all the decoupled optimization problems, which represents $S \cdot N$ solving agents. Followed by equality-constrained coupled/consensus QP

$$\begin{aligned} \min_{v^+, z^+} \; & \sum_{k=0}^{N-1} \left\| \begin{bmatrix} z^+(k) - x(k) \\ v^+(k) - u(k) \end{bmatrix} \right\|_\Omega^2 + \begin{bmatrix} \nabla f_x \\ \nabla f_u \end{bmatrix}^\top \begin{bmatrix} z^+(k) - x(k) \\ v^+(k) - u(k) \end{bmatrix} \\ & + \|z(N)^+ - x(N)\|_P^2, \\ \text{s.t. } & \forall k \in \{0, \dots, N-1\}, \\ & z^+(k+1) = Az^+(k) + Bv^+(k) \quad |\lambda^+(k), \\ & C \left[z^+(k) - x(k), \quad v^+(k) - u(k) \right]^\top = 0, \\ & z(0)^+ = x(0), \end{aligned} \tag{3}$$

where symmetric positive definite matrix $\Omega \in \mathbb{R}^{n_\Omega \times n_\Omega}$ consists of weighting matrices Q and R placed on its diagonal, $\nabla f_\mathrm{x} \in \mathbb{R}^{n_x}$ and $\nabla f_\mathrm{u} \in \mathbb{R}^{n_u}$ represents gradient of the original problem in (1) with respect to x and u respectively. $C \in \mathbb{R}^{n_\mathcal{A} \times n_c}$ is constructed as a matrix corresponding to an active set of constraints $\mathcal{A}$ from all decoupled QPs (2), where $n_\mathcal{A}$ represents the number of inequality constraints from

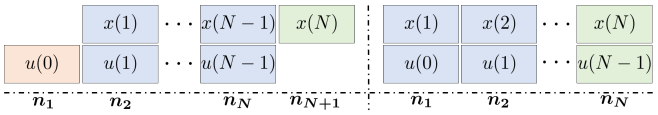

Fig. 1. The distribution of MPC problem: original method in [15] (left), proposed method (right).

original problem (1). The λ^+ is a dual variable associated with coupled equality constrain representing the model of the controlled system. After computing the primal-dual solution of the consensus problem algorithm updates variables $z \leftarrow z^+$, $v \leftarrow v^+$ and $\lambda \leftarrow \lambda^+$, see [9].

Through iterative solution of the decoupled QP in (2) and coupled QP in (3), with variable updates (z, v, λ), the optimal trajectories of $x^\star$ and $u^\star$ for the original MPC design problem in (1) are constructed.

Execution of an iterative algorithm can still require significant computational power and time because it solves multiple constrained QPs in each iteration. In order to reduce the burden of real-time computation, it is possible to obtain an explicit solution for the decoupled QPs in (2). This approach is independent of the length of the prediction horizon, as noted in [15]. Nevertheless, the issue of dimensionality remains since the total number of non-distributed parameters for n_{x_i} and n_{u_i} must not exceed a specific limit to enable the construction and real-time computation of an explicit solution on hardware-limited memory.

III. Deep-Learning-Driven Parallel MPC

In general, a multi-parametric QP (mpQP) for decoupled QPs in (2) is formulated as follows

$$U_i^\star(\theta_i) = \underset{u_i(k), x_i(k+1)}{\arg\min} \; J_{\mathrm{D}}(u_i(k), x_i(k+1)), \tag{4}$$

where the explicit representation $U_i^\star(\theta)$ is piece-wise affine (PWA) function [23], [24]. Such PWA control law is defined by the vector for parameters $\theta_i = \left[z^\top(k), z_i^\top(k+1), v^\top(k), \lambda_i^\top(k+1)\right]^\top \in \mathbb{R}^{n_{\theta_i}}$, where n_{θ_i} represents the number of all component from states and control coupling vectors, and Lagrange multipliers of coupling constraints, respectively. The aim is to find an approximated solution for the decoupled QPs in (4) such that holds

$$\hat{U}_i(\theta) \approx U_i^\star(\theta), \tag{5}$$

where $\hat{U}_i(\theta)$ represents a desired approximated solution. To efficiently address the concerns raised in the preceding section, we implement a deep-learning-based approximation of (2).

A. Deep-learning-driven approximation of control law

Deep learning algorithms or deep neural networks (DNN) are widely known for their capability of learning nonlinear patterns from data [25]. Trained DNNs are general nonlinear functions that can fit any dynamics and represent parametric optimization problems. Various strategies have been introduced in the literature on utilizing DNNs as a parametric solution of MPC [26], [27]. Several architectural options are available within the framework of DNNs, among others, the recurrent neural networks well-suited for handling sequential data [28], or autoencoders effectively reducing the dimensionality [29]. Each DNN carries its own set of additional advantages. However, not all DNN structures are well suited to solve problems defined in Section II-B, as it is important to compromise between structural complexity and performance loss. A particularly suitable choice is the multi-layer neural network due to its capacity for learning complex nonlinear dynamics while composed of conventional neurons.

The proposed real-time control method is based on the perspective approach of mimicking the solution of the decoupled QPs in (2) on the distributed layer. DNN is trained to map the input parameters onto the approximate solution $\hat{u}_i$ value while minimizing the perturbations subject to the optimal reference solution $u_i^\star$. Results are achieved by solving the regression optimization problem in the form of minimizing the sum-of-squared-error-based criterion (SSE)

$$\min_{\gamma} \quad \sum_{m=1}^{M} \left(Y_m - \hat{Y}\right)^2, \quad \text{s.t.: } \hat{Y} = f_{\mathrm{DNN}}(\theta_m, \gamma), \tag{6}$$

where f_{DNN} is fully connected neural network model, consists of H neurons arranged in multiple layers L with predefine activation function σ. By solving the (6), optimal weights and biases γ of DNN are evaluated. The training dataset $\mathcal{D} = \{Y_m, \theta_m\}_m^M$ includes optimal control trajectories $u_m^\star = Y_m$, with define DNN inputs θ_m corresponding to parameters of parametric optimization (4).

The proposed DNN represents a parametric function in the following form

$$\hat{u}_i(k) = f_{\mathrm{DNN}}(\theta_i(k)), \tag{7}$$

where $\hat{u}_i(k)$ is an output of the neural network approximating the optimal control law. To reduce neural network complexity, the state vector is not present as one of the DNN outputs. However, by additional computation, the approximate state vector can be obtained

$$\hat{x}(k+1) = Ax(k) + B\hat{u}(k). \tag{8}$$

Combining both approximated parametric solution can be found $\hat{U}_i(\theta) = [\hat{u}_i^\top(k), \hat{x}_i^\top(k+1)]^\top$ that corresponds to original mpQP in (4).

In various cases, methods such as mimicking dynamics may not be adequate without a unique architecture or training of DNNs. Nevertheless, to implement a real-time ALADIN algorithm, the DNN model is preferred to be simple, even if it sacrifices the accuracy of an imprecise parametric solution. The coordination layer of the consensus problem in (3) allows the ALADIN algorithm efficiently solve complex optimization problems. Simultaneously, it maintains the error between the approximate decoupled layers and the original equality-constrained quadratic programming (QP) problem by minimizing the difference $z_i^+(k) - \hat{x}_i(k)$ and $v_i^+(k) - \hat{u}_i(k)$. In other words, the coupling QP problem represents a compensator that corrects the DNNs output in the desired direction. However, it is essential to ensure that trained DNN always converges in a finite number of iterations towards the neighborhood of the optimal solution enabling the coupling QP to provide stability by use for proposed ALADIN-NN-

based MPC. This property can be achieved by design by providing a large number of sufficiently training datasets.

B. Implementation Details

To properly tune the training of the DNN, it is necessary to generate training and testing datasets. This can be realized, for instance, by using the parametric solution of the decoupled QP in (2) or by repeatability solving the ALADIN method according to Section II-B. We advise a second option as the preferred one, as it is applicable in any circumstances. However, many parametric solvers provide a variety of data generations, including the Chebyshev center or equidistant griding of feasible parametric space. However, all these benefits of parametric solvers are restricted to having a low number of parameters as the mentioned methods are not scalable. For large-scale systems, the only mandatory condition is to cover a reasonable portion of investigated parametric space with various combinations [30].

As shown in Section II-B, nominal MPC was distributed to N symmetric subproblems. Resulting in the two unique parametric solutions (4) for each space distribution $i = 1, \dots, S$. First explicit solution of $J_{\mathrm{D}}(u_i(N-1), x_i(N))$ is associated with terminal set, and second explicit solution of $J_{\mathrm{D}}(u_i(k), x_i(k+1))$ is associated with the $k = \{0, \dots, N-2\}$ steps of the prediction horizon. As the proposed method mimics the exact parametric solution of decoupled QP in (2), it is needed to train $2 \cdot S$ different neural networks where $T_i(\theta)$ represents a parametric solution of $J_{\mathrm{D}}(u_i(N-1), x_i(N))$ and $R_i(\theta)$ represents a parametric solution of $J_{\mathrm{D}}(u_i(k), x_i(k+1))$.

Algorithm 1 shows the proposed real-time deep-learning-driven parallel MPC scheme utilizing the ALADIN algorithm and DNN. The main difference from the original ALADIN algorithm proposed in [9] is in a parametric approximation of decoupled inequality-constrained QP. The provided implementation significantly reduces overall online computational time. Compared to the exact explicit solution in [12], utilizing approximate DNN parametrization enables implementation for large-scale systems and low-level hardware implementation by reducing the memory footprint. The downside is the lost convergence guarantees and optimality of control trajectories. As the trained DNNs are only approximations of parametric solutions of mpQP in (4), they can occasionally exceed constraints that are then post-processed by saturation in Step 14. On the other hand, if produced approximation of control action $\hat{u}$ is close to the optimum and is inside of the constraints, it is corrected by coupled equality constrained QP (3) ensuring feasible, but possibly suboptimal control trajectory $\overline{u}$. Same as in the original algorithm number of ALADIN iterations a can be affected by initial guesses in Step 3 in the online phase, but it can be warm-started for $a > 1$ with the previous MPC solution. As a hard-stopping criterion, one can choose a particularly small number of $a_{\max}$ as, in practice, well-trained DNNs converge near the optimal, as fast as an exact solution with the same initial guesses. However, this method is missing rigorous guarantees. In addition, the proposed Algorithm 1 is modified to include stabilizing LQR, i.e., the well-known Dual Mode Control ensures stability, once the measured states $x(t)$ are in the terminal set $\mathbb{X}_{\mathrm{P}}$. Moreover, it is possible to ensure the stability of deep-learning-driven MPC directly by introducing the principles of tube MPC, see [31].

Algorithm: Deep-Learning-Driven Parallel MPC

Offline Phase:

1: Generate dataset $\mathcal{D} = \{u_m^\star, \theta_m\}_m^M$ by repeatability solving ALADIN (Section II-B) in closed-loop simulation for different initial conditions $x(0) = x_t$.
2: Train $2 \cdot S$ deep neural networks $\hat{u}_i(k) = f_{\mathrm{DNN}}(\theta_i(k))$ parametrise as explicit solution of decoupled optimisation problem (2) to construct approximated control law $\hat{u}(k) \approx u^\star(k)$.

Online Phase:

3: Initial guesses of $z \in \mathbb{R}^{n_x \times N}$, $v \in \mathbb{R}^{n_u \times N}$, $\lambda \in \mathbb{R}^{n_x \times N}$.
4: Set $x(0) \leftarrow x(t)$.
5: **for** $a = 1, \dots, a_{\max}$ **do**
Solve in parallel:
6: **for** $k = 0, \dots, N-1$ and $i = 1, \dots, S$ **do**
7: Select $z_i(k)$, $v_i(k)$, $\lambda_i(k)$.
8: $\theta_i(k) \leftarrow \left[z_i^\top(k), z_i^\top(k+1), v_i^\top(k), \lambda_i^\top(k+1)\right]^\top$.
9: **if** $k = N-1$ **then**
10: Set $\hat{u}_i(N-1) \leftarrow T_i(\theta_i(N-1))$.
11: **else**
12: Set $\hat{u}_i(k) \leftarrow R_i(\theta_i(k))$.
13: **end if**
14: Saturate $\hat{u}_i(k) = \min(u_{\max,i}, \max(u_{\min,i}, \hat{u}_i(k)))$.
15: **end for**
Consensus problem:
16: Set $\hat{x}(k+1) \leftarrow Ax(k) + B\hat{u}(k)$.
17: Solve QP in (3) for $\hat{u}_i(k)$ and $\hat{x}_i(k+1)$ for all $i \in \{1, \dots S\}$ and $k \in \{0, \dots, N-1\}$.
18: **if** $\|v(k) - v^+(k)\|_\infty \leq \mathrm{TOL}$ **then**
19: **break**.
20: **end if**
21: Set $z \leftarrow z^+$, $v \leftarrow v^+$, $\lambda \leftarrow \lambda^+$.
22: **end for**
23: Set $\overline{u} \leftarrow v$ as inexact control law.
24: Apply $\overline{u}(0)$ to the real process.
25: Wait for the state measurement $x(t)$ and go to Step 3.

IV. Case Study

In this numerical study, we introduce a large-scale system to demonstrate the capabilities of the proposed deep-learning-driven parallel MPC and its ability to be implemented in low-level hardware. To assess the algorithm's elapsed time and memory footprint, we perform a simulation over the benchmark system and compare it to other established methods for solving the optimization problem. Although the investigated system can solve by nominal MPC within the considered sampling time, it is necessary to utilize a commercially licensed solver, e.g., `Gurobi` or `Mosek`, to

name a few. On the other hand, the runtimes of the freely available solvers such as `CVXOPT`, `ECOS`, or `Quadprog` were significantly slower or even inapplicable to solve the MPC problem. However, the proposed deep-learning-driven parallel MPC approach has none of the above-listed limitations, as it solves simple equality-constrained quadratic programs.

A. The reactor-separator plant

The case study was performed on the benchmark system adopted from [32]. The process consists of three-vessel, two continuously stirred tank reactors, and a tank separator. Two reactions are ongoing in both reactors. The first one is the conversion of the reactant A into the main product B. The second reaction is the undesired conversion of the product A into the side-product C. The reactant A enters the first reactor in the feed stream F_{10}. The outlet of the first reactor F_1 is mixed with additional feed containing reactant A F_{20} to make the inlet to the second reactor. The products of the reactions in the second reactor are sent to the separator, from which the vapors are condensed and recycled to the first reactor in the stream F_{r}. The bottom product is then removed in the stream F_3.

For the purposes of the case study, the reactor-separator plant is divided into three subsystems: the reactor #1, the reactor #2, and the separator #3. For clarity, these subsystems are referred to using subindex $i \in \{1, 2, 3\}$. Since the process has one unstable and two stable steady-state values [32], the control aims to steer the state variables: temperatures (T_i) and concentrations of the reactant A, and products B and C: $(c_{\mathrm{A},i}, c_{\mathrm{B},i}, c_{\mathrm{C},i})$ to the optimal setpoint represented by the unstable steady-state. In the case study, the control inputs are the heat flows: H_1, H_2, and H_3, representing the external heat delivered or removed from the tanks. The steady-state values of system state variables and control inputs for all subsystems are adopted from [32].

The MPC is tuned with respect to the constraints $\left[-0.5, -1.5, -2\right]^\top \cdot 10^5 \leq u \leq \left[0.5, 1.5, 2\right]^\top \cdot 10^5, -x^s \leq x,$ where presented variables, exception system steady-states x^s, and constraints values are stated in the deviation form: the state variables x contain the deviation from their steady-states of the variables T_i, $c_{\mathrm{A},i}, c_{\mathrm{B},i}, c_{\mathrm{C},i}$ for all three subsystems. The control input u is also comprised of heat flows (H_1, H_2, H_3) in the deviation form subject to the corresponding steady-state values. The detailed description of the reactor-separator plant and the derivation of the mathematical model describing the dynamics in all three devices is given in [32].

B. Implementation and control and setup

The MPC setup is given for prediction horizon $N = 15$, and sampling time $T_{\mathrm{s}} = 0.1$ h, weighting matrices are defined as $R = \mathrm{diag}(0.01, 0.01, 0.01), Q = \mathrm{diag}(Q_1, Q_2, Q_3)$, where $Q_i = \mathrm{diag}(3200, 1, 1, 1) \cdot 10^3$ for $i = 1, 2, 3$, and P is computed by solving an algebraic matrix Riccati equation.

The ALADIN algorithm distributed nominal MPC into the 45 decoupled optimization problems with scaling parameter $\rho = 1 \cdot 10^6$. To minimize the real-time burden, the stopping criterion was selected as $a_{\max} = 1$. This particular choice of aggressive early stooping criteria is introduced to show the fast convergence rate of the proposed algorithm that operates near the optimal trajectories by observation. In practice, one can choose $a_{\max} > 1$ to achieve desired suboptimality level, see [13].

As a benchmark, Algorithm 1 was implemented in `Python 3.8` using the `PICOS` library and `Gurobi` solver. Training and testing dataset $\mathcal{D}$ were generated by performing closed-loop simulations with ALADIN algorithm (Section III-B) producing $M = 184\,114$ individual samples. The large number of generated samples results in a notable computational burden during the offline phase. Note, that this paper does not incorporate the minimum sample evaluation that retains equivalent performance to the chosen dataset. This aspect will be addressed in future work.

The created dataset was then divided by the ratio $70 : 15 : 15$ representing training, testing, and validation data, respectively, followed by min-max normalization to scale and improve DNN training. The found DNN-based approximation of parametric solution consisted of $L = 7$ hidden layers, each with $H = 95$ neurons and activation function as $\sigma = \mathrm{ReLU}()$ representing the rectified linear unit. The parameterization of the decoupled layer θ_i includes 22 parameters, which are also inputs to DNN. The training procedure was performed by ADAM optimizer with the learning rate of $6.11 \cdot 10^{-6}$, and $3\,870$ epochs were found sufficient to train DNN accurately. As the decomposition of nominal MPC results in symmetric decoupled optimization problems, we applied exactly the same training procedure for each parametric solution, producing $R_i(\theta)$ and $T_i(\theta)$ neural networks with similar accuracy for each space distribution $i = \{1, \ldots, 3\}$. The demonstrated single neural network comprises $57\,001$ floating-point parameters, corresponding to approximately 222.7 kB of hardware storage. This motivates a practical choice for deployment in low-level hardware such as ESP32-S3, which has a static random-access memory (SRAM) capacity ranging from 520 kB to 8 MB with external flash/SRAM memory.

C. Results and Discussion

To demonstrate the performance of the proposed Algorithm 1, the closed-loop simulations were performed, analyzing both the steady-state stabilization and disturbance rejection control problems. The simulation starts with the initial condition $x_1(0) = [-8.84, -0.12, -0.02, -0.01]^\top$, $x_2(0) = [-4.34, 0.01, -0.11, -0.03]^\top$ and $x_3(0) = [-4.83, -0.09, -0.12, -0.04]^\top$, where values of $x_{i,1}, \ldots, x_{i,4}$ represent the deviation variable for the temperature and concentrations of the reactant A and products B and C in the i-th subsystem.

After the system reaches desired steady-state in $t \approx 1$ h, we generate external temperature disturbance $\delta = 15$ K for $t = 1$ h in each process. As can be seen in Figures 2, all control strategies perform almost identically. The nominal MPC serves as a basis for the other two approaches, we

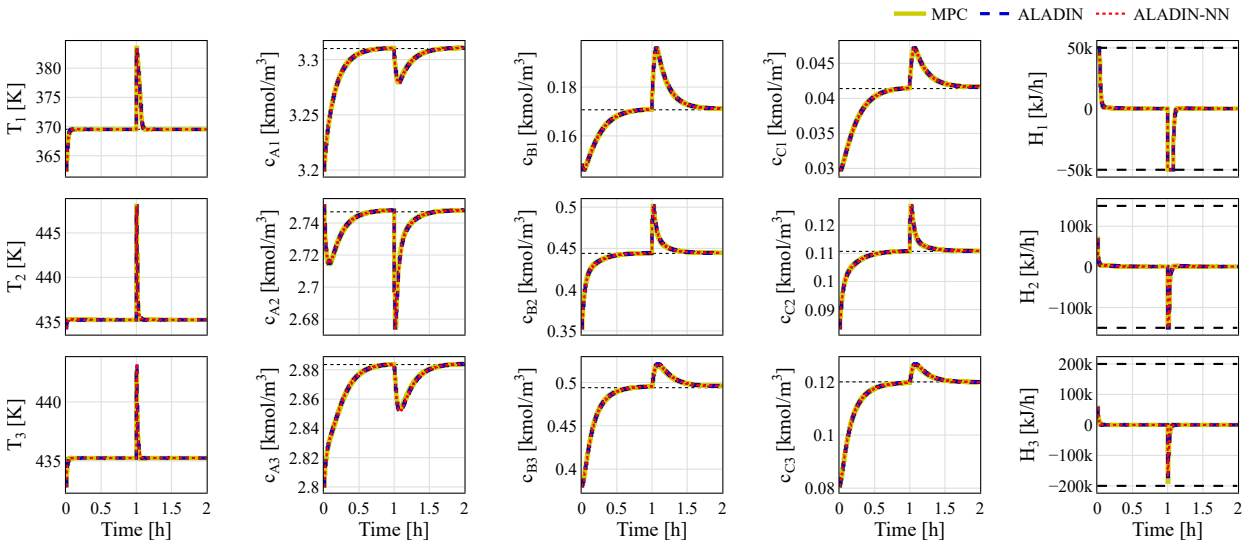

Fig. 2. From the left first four columns represents the system state control trajectories of closed-loop simulation: nominal MPC (yellow), rigorous ALADIN-based MPC (blue), proposed ALADIN-NN-based MPC (red), and steady-state reference x^s (black). The last column show closed-loop control profiles of the control inputs: nominal MPC (yellow), rigorous ALADIN-based MPC (blue), proposed ALADIN-NN-based MPC (red), and constraints (black).

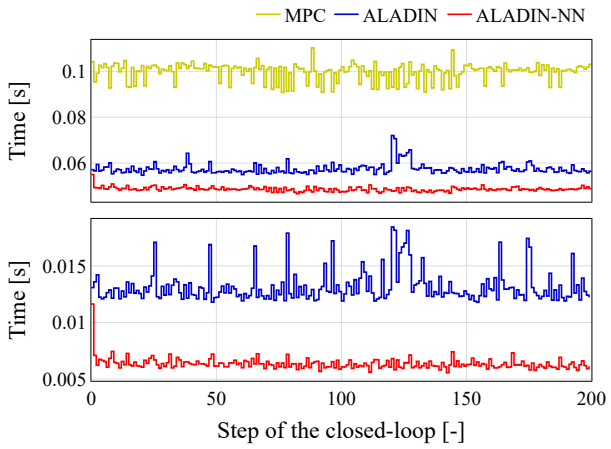

Fig. 3. Computational complexity of evaluated MPC design methods in closed-loop simulation: (i) upper plot: overall runtime, (ii) bottom plot: parallel elapsed time of the decoupled subproblems.

TABLE I

OVERALL RESULTS OF CLOSED-LOOP SIMULATION.

Method	t_{tot} [s]	$\bar{t}$ [ms]	$t_{\max}$ [ms]	t_{c} [ms]	$\widetilde{J}$ [%]
MPC	19.95	99	110	20	0.0
ALADIN	11.49	57	72	1 + 4	1.18
ALADIN–NN	9.72	49	55	1	1.18

observe suboptimal behavior from both controllers, mainly during the disturbance rejection where $t > 100$. To measure the rate of suboptimality, we use the following formula

$$\widetilde{J} = \sum_{\tau=1}^{200} \left(\frac{J_\tau(\overline{x}, \overline{u}) - J_\tau(x^\star, u^\star)}{J_\tau(x^\star, u^\star)} \right) \cdot \frac{100\%}{200}, \qquad (9)$$

where J_τ represents objective function (1) after the solution is found, τ stands for step of the closed-loop.Here, $x^\star$ and $u^\star$ denote optimal state and control trajectories from nominal MPC, $\overline{x}$ and $\overline{u}$, respectively, represent approximate state and control trajectories generated from the ALADIN algorithm and proposed Algorithm 1. Table I summarizes the overall simulation results. As $\widetilde{J}$ indicates, both ALADIN-based MPC design methods provide control profiles close to optimum. The numbers differ after the five decimal points, which is negligible. The main difference between compared methods is the computational burden. As the upper graph of Figure 3 indicates, the worst-case runtime across all steps of closed-loop simulation has naturally nominal MPC (Figure 3, green). The ALADIN algorithm (blue line) shows that time and space distribution reduced the optimization complexity leading to the decreased total elapsed time t_{tot}. Furthermore, through the approximate parameterization of parallel MPC (Figure 3, red), we attained the best outcome regarding computational complexity. Table I presents a detailed time domain analysis, encompassing the mean $\bar{t}$, total t_{tot}, and worst case $t_{\max}$ elapsed times. These values are calculated by averaging the results obtained from ten runs of the same simulation. The statistical data supports the assertion that adopting a deep-learning-driven parametrization of the decoupled optimization problem (Algorithm 1) results in equivalent accuracy to original ALADIN methods with improved computational performance. The elapsed time

difference can be observed in the lower graph of Figure 3, with almost $50\,\%$ time reduction. The hidden advantage of the proposed method is that we only construct a coupled optimization problem in each iteration, and the decoupled optimization is solved with the aid of DNN. Table I displays the average construction time t_{c} for each method. However, this variable is specific to the used programming language (`Python 3.8`) and the optimization constructor (`PICOS`). Based on the results provided, we can conclude that the proposed method has a construction time t_{c} that is reduced by a factor of 20 compared to the nominal MPC and is in $4\,\mathrm{ms}$ faster than the original ALADIN algorithm.

V. Conclusion

The paper presented an advanced process control using a novel real-time control policy based on a deep-learning approximation of the distributed optimization solution of the ALADIN-based parallel MPC design framework. Results of the numerical study presented on a benchmark system prove the suitability of the proposed algorithm for real-time control even for more complex systems. A comparison of the elapsed time in the closed-loop simulation showed almost $50\,\%$ time reduction with the deep-learning-based approach compared to the nominal MPC. Furthermore, the proposed algorithm is suitable for execution on low-level hardware with ease, and it has the potential for great scaling with both the prediction horizon and the state space dimension.

Since the algorithm presented in this paper is based on the ALADIN method designed for solving nonlinear MPC problems [5], future research will include an investigation of the possibility of implementing the proposed control policy towards nonlinear real-time parallel MPC design.

References

[1] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: an engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, pp. 1327–1349, 2021.

[2] R. Oberdieck, N. D. A., I. Nascu, M. P. M, M. Sun, S. Avraamidou, and E. N. Pistikopoulos, "On multi-parametric programming and its applications in process systems engineering," *Chemical engineering research and design*, vol. 116, pp. 61–82, 2016.

[3] A. Bemporad, M. Morari, V. Dua, and E. P. N., "The Explicit Linear Quadratic Regulator for Constrained Systems," *Automatica*, vol. 38, pp. 3–20, 1999.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine learning*, vol. 3, no. 1, pp. 1–122, 2011.

[5] B. Houska and J. Shi, "Distributed MPC with ALADIN–a tutorial," in *2022 American Control Conference (ACC)*, pp. 358–363, 2022.

[6] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, "Recent advances in quadratic programming algorithms for nonlinear model predictive control," *Vietnam J. Math*, vol. 46, pp. 863–882, 2018.

[7] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: an operator splitting solver for quadratic programs," *Mathematical Programming Computation*, vol. 12, no. 4, pp. 637–672, 2020.

[8] A. Engelmann, Y. Jiang, H. Benner, R. Ou, B. Houska, and T. Faulwasser, "ALADIN-α–an open-source matlab toolbox for distributed non-convex optimization," *Optimal Control Applications and Methods*, vol. 43, no. 1, pp. 4–22, 2021.

[9] B. Houska, J. Frasch, and M. Diehl, "An augmented lagrangian based algorithm for distributed nonconvex optimization," *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1101–1127, 2016.

[10] A. Kozma, J. F. V., and M. Diehl, "A distributed method for convex quadratic programming problems arising in optimal control of distributed systems," in *52nd IEEE Conference on Decision and Control*, pp. 1526–1531, IEEE, 2013.

[11] S. Richter, M. Morari, and C. J. N., "Towards computational complexity certification for constrained MPC based on lagrange relaxation and the fast gradient method," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*, pp. 5223–5229, IEEE, 2011.

[12] J. Oravec, Y. Jiang, B. Houska, and M. Kvasnica, "Parallel explicit MPC for hardware with limited memory," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 3301–3306, 2017.

[13] Y. Jiang, J. Oravec, B. Houska, and M. Kvasnica, "Parallel MPC for linear systems with input constraints," *IEEE Transactions on Automatic Control*, vol. 66, pp. 3401–3408, July 2021 2021.

[14] K. Wang, Y. Jiang, J. Oravec, M. Villanueva, and E. B. Houska, "Parallel explicit tube model predictive control," in *58th IEEE Conference on Decision and Control*, 58, (Nice, France), pp. 7696–7701, December 11-13, 2019 2019.

[15] J. Shi, Y. Jiang, J. Oravec, and B. Houska, "Parallel MPC for linear systems with state and input constraints," *IEEE Control Systems Letters*, vol. 7, pp. 229–234, 2022.

[16] A. Shokry and E. Moulines, "Health-constrained explicit model predictive control based on deep-neural networks applied to real-time charging of batteries," *IFAC-PapersOnLine*, vol. 55, no. 16, pp. 142–147, 2022.

[17] Y. Cho, G. Hwang, D. G. Quarme, and S. Hwang, "Artificial neural network-based model predictive control for optimal operating conditions in proton exchange membrane fuel cells," *Journal of Cleaner Production*, vol. 380, p. 135049, 2022.

[18] S. Grosébastien and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Transactions on Automatic Control*, vol. 65, no. 2, pp. 636–648, 2019.

[19] D. Masti and A. Bemporad, "Learning binary warm starts for multiparametric mixed-integer quadratic programming," in *2019 18th European Control Conference (ECC)*, pp. 1494–1499, IEEE, 2019.

[20] W. Jia, S. Qin, and X. Xue, "A generalized neural network for distributed nonsmooth optimization with inequality constraint," *Neural Networks*, vol. 119, pp. 46–56, 2019.

[21] H. Xiao, C. C. Philip, G. Lai, D. Yu, and Y. Zhang, "Integrated nonholonomic multi-robot consensus tracking formation using neural-network-optimized distributed model predictive control strategy," *Neurocomputing*, vol. 518, pp. 282–293, 2023.

[22] J. Rawlings and D. Mayne, "Postface to model predictive control: Theory and design," *Nob Hill Pub*, vol. 5, pp. 155–158, 2012.

[23] A. Bemporad, M. Morari, V. Dua, and E. P. N, "The explicit linear quadratic regulator for constrained systems," *Automatica*, vol. 38, no. 1, pp. 3–20, 2002.

[24] F. Borrelli, *Constrained optimal control of linear and hybrid systems*, vol. 290. Springer, 2003.

[25] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse problems*, vol. 34, no. 1, p. 014004, 2017.

[26] S. Chen, K. Saulnier, N. Atanasov, D. L. D, V. Kumar, G. P. J, and M. Morari, "Approximating explicit model predictive control using constrained neural networks," in *2018 Annual American control conference (ACC)*, pp. 1520–1527, IEEE, 2018.

[27] E. M. Tanowe, C. M. da S, G. Waltrich, and C. J. N, "A neural network architecture to learn explicit MPC controllers from data," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 11362–11367, 2020.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[29] D. Masti and A. Bemporad, "Learning nonlinear state–space models using autoencoders," *Automatica*, vol. 129, p. 109666, 2021.

[30] K. Liu and A. Belletélien, "Escaping the curse of dimensionality in similarity learning: Efficient frank-wolfe algorithm and generalization bounds," *Neurocomputing*, vol. 333, pp. 185–199, 2019.

[31] A. Aswani, H. Gonzalez, S. S. Shankar, and C. Tomlin, "Provably safe and robust learning-based model predictive control," *Automatica*, vol. 49, no. 5, pp. 1216–1226, 2013.

[32] G. Lars, A. Frank, R. Findeisen, J. Fischer, D. Groß, U. D. Hanebeck, B. Kern, M. Müller, J. Pannek, and M. Reble, "Distributed and networked model predictive control," *Control Theory of Digitally Networked Dynamic Systems*, pp. 111–167, 2014.