This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

A Practical Multiobjective Learning Framework for Optimal Hardware–Software Co-Design of Control-on-a-Chip Systems

Kimberly J. Chan, *Graduate Student Member, IEEE*, Joel A. Paulson, and Ali Mesbah, *Senior Member, IEEE*

***Abstract*— The digital age has made embedded control a key component to user-oriented, portable, and the Internet of Things (IoT) devices. In addition, with emergent complex systems, there is a need for advanced optimization-based control strategies such as model predictive control (MPC). However, the unified implementation of these advanced strategies on hardware remains a challenge. Designing complex control policies for embedded systems is inherently an interwoven process between the algorithmic design and hardware implementation, which will require a hardware–software co-design perspective. We propose an end-to-end framework for the automated design and tuning of arbitrary control policies on arbitrary hardware. The proposed framework relies on deep learning as a universal control policy representation and multiobjective Bayesian optimization (MOBO) to facilitate iterative systematic controller design. The large representation power of deep learning and its ability to decouple hardware and software design are a central component to determining feasible control-on-a-chip (CoC) policies. Then, Bayesian optimization (BO) provides a flexible sequential decision-making framework where practical considerations, such as multiobjective optimization (MOO) concepts and categorical decisions, can be incorporated to efficiently design embedded control policies that are directly implemented on hardware. We demonstrate the proposed framework via closed-loop simulations and real-time experiments on an atmospheric pressure plasma jet (APPJ) for plasma processing of biomaterials.**

***Index Terms*— Cold atmospheric plasmas (CAPs), embedded control, hardware–software co-design, imitation learning, multiobjective Bayesian optimization (MOBO).**

I. Introduction

EMBEDDED systems lie at the core of many online control systems technologies, including autonomous systems [1], Internet of Things (IoT) systems [2], [3], and biomedical devices [4], [5], among others. Microcontrollers/microprocessors (MCUs) have played a major role in enabling embedded control for portable devices [6]. Due to their widespread availability and adoption, MCUs have been designed such that they can be easily programmed using high-level programming languages, such as C and Python, and can be deployed with relative ease [2]. Meanwhile, emerging technologies have increasingly complex dynamics and typically rely on advanced model-based control strategies, such as model predictive control (MPC), that can handle constraints. As a result, significant efforts have targeted the development of automated software-based code generation tools for fast numerical optimization on MCUs. These tools (e.g., ACADO [7], GRAMPC [8], and FORCES [9]) utilize tailored implementations of structured formulations of the underlying optimization problem to efficiently compute the optimization solution for real-time control (e.g., using sequential quadratic programming [10], nonlinear interior point methods [11], or accelerated gradient methods [12]). The structured formulation that these code generation tools require can make the implementation of robust and learning-based optimal control strategies more challenging, while the focus on solely the software side implies that the form of hardware implementation is also limited. Furthermore, the design of embedded controllers relies on more than just the fast solution of the optimal control problem. It also involves maintaining numerical robustness at low computational accuracy, tolerance against infeasibility, low code complexity, and low memory/resource utilization, among other considerations. Many of these challenges require explicit knowledge of the hardware specifications, as well as knowledge of how computations are performed and accelerated on the hardware. As advanced hardware technologies [e.g., graphics processing units (GPUs), field-programmable gate arrays (FPGAs), and tensor processing units (TPUs)] become commonplace, the principle of *hardware–software co-design* will become an integral consideration for the physical implementation of embedded controllers [13]. Hardware–software co-design involves the concurrent design of the control algorithm (software) and its embedded implementation (hardware) [14].

Optimized control policy tuning is often a tedious and cumbersome process, which is further exacerbated by the extensive workflow to go from the programmatic control policy design to embedded implementation. Control policy auto-tuning (also known as calibration) is well established for simple controllers [15], [16], but auto-tuning for generic control structures has recently regained traction. One popular approach to auto-tuning uses principles of reinforcement

Manuscript received 13 November 2023; revised 9 March 2024; accepted 14 May 2024. This work was supported by the U.S. National Science Foundation under Grant 2317629 and Grant 2029282. Recommended by Associate Editor C. N. Jones. *(Corresponding author: Ali Mesbah.)*

Kimberly J. Chan and Ali Mesbah are with the Department of Chemical and Biomolecular Engineering, University of California at Berkeley, Berkeley, CA 94720 USA (e-mail: kchan45@berkeley.edu; mesbah@berkeley.edu).

Joel A. Paulson is with the Department of Chemical and Biomolecular Engineering, The Ohio State University, Columbus, OH 43210 USA (e-mail: paulson.82@osu.edu).

Digital Object Identifier 10.1109/TCST.2024.3407582

1063-6536 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

learning (RL) [17], [18], [19]. To this end, policy-gradient RL methods involve updating the control policy parameters via gradient descent [20]. While policy gradient is a scalable approach, it can require many evaluations on the true system and can be prone to getting stuck at local optimizers. Another popular approach to auto-tuning relies on data-driven optimization, particularly Bayesian optimization (BO) [21], [22], [23]. Auto-tuning can be interpreted as a black-box problem where the objective function is expensive to evaluate, potentially nonconvex, and without closed-form derivatives. BO is a "global" optimization method that takes a probabilistically principled approach to reduce the number of interactions with the real system [24]. Variants of BO for auto-tuning are also emerging [25], [26], [27]; however, most of these studies, except [26], do not consider the hardware considerations for embedded control. Even still, the focus of [26] remains on the hardware-constrained optimization (i.e., real-time computations) of the software, rather than hardware–software co-design.

The hardware–software co-design paradigm presents a new take on the embedded control design problem. Formally, we denote this new perspective of hardware–software co-design as control-on-a-chip (CoC) design since we aim to provide a unified workflow to place arbitrary control policies on arbitrary hardware. In this work, we pose the CoC co-design problem as an optimization problem that incorporates a hardware feasibility constraint and multiple levels of decisions to create an all-in-one framework. First, we establish a flexible workflow that takes advantage of recent advances in deep learning. Deep neural networks (DNNs) have played a pivotal role in imitation learning of MPC policies [28], [29], [30] and differentiable predictive control [31]. In this work, we use the imitation learning perspective as it poses two key advantages: 1) it provides a "physically interpretable" control policy and 2) by virtue of 1), it reduces the design parameter space. Thus, our proposed CoC workflow consists in: 1) designing an expert control policy (based on optimization-based control strategies); 2) using DNNs to represent the expert control policy; and 3) implementing the DNN-based control policy on hardware. We then use multiobjective BO (MOBO) [32] to encapsulate the multistep CoC design workflow to create an end-to-end optimization framework. Solving a multiobjective CoC design problem via MOBO entails finding an optimal set of control policies rather than one single optimizer, which allows a practitioner to choose the best design(s) according to the needs or preferences of the application [33], [34]. Furthermore, BO offers flexibility when incorporating design choices from each step of the CoC workflow [35], [36].

This article is organized as follows. Section II presents a mathematical foundation for the optimal CoC design framework. Section III describes the role of deep learning as our universal control policy representation. The unique advantages of MOBO for an end-to-end CoC design are detailed in Section IV. Section V demonstrates the key advantages of imitation learning via deep learning using an illustrative example. In Section VI, the proposed framework is applied to the CoC design for an atmospheric pressure plasma jet (APPJ) with prototypical biomedical applications. Section VII concludes this article.

II. Problem Formulation

In this work, we seek optimal CoC policies that minimize a set of closed-loop cost metrics subject to hardware constraints. In general, CoC policies can be represented as a space of all possible machine code instructions that can be executed on a given choice of hardware. Since this is a complex design space involving decisions made by human experts, the design problem is often decomposed into two key steps: 1) choice of a high-level control program θ that must reside in the space of possible programs Θ (generally very high-dimensional and complex) and 2) choice of a code generation strategy that translates θ into executable machine code, which has its own set of design parameters denoted by $\gamma \in \Gamma$ (e.g., numerical representation and parallelization options). We can formulate the search for an optimal pair of program and code generation parameters ($\theta^\star$ and $\gamma^\star$) in terms of the following multiobjective optimization (MOO) problem:

$$\min_{\theta,\gamma} \ \{J_1, \ldots, J_M\} \tag{1a}$$

$$\text{s.t. } x_{t+1} = f(t, x_t, u_t, w_t) \tag{1b}$$

$$u_t = \pi(x_t; \theta, \gamma) \tag{1c}$$

$$w_t \sim P_{w_t}(x_t, u_t) \tag{1d}$$

$$J_i = \mathbb{E}\left\{ \ell_{T,i}(x_T) + \sum_{t=0}^{T-1} \ell_i(x_t, u_t, w_t) \right\} \tag{1e}$$

$$g(\theta, \gamma) = 1 \tag{1f}$$

$$(\theta, \gamma) \in \Theta \times \Gamma$$
$$\forall (t, i) \in \{0, \ldots, T-1\} \times \{1, \ldots, M\} \tag{1g}$$

where (1a) represents the set of M closed-loop performance metrics; (1b) represents the system dynamics that describes the evolution of the system state x_t in response to control actions u_t and disturbances w_t at time t; (1c) defines the CoC policy $\pi(x_t; \theta, \gamma)$ that maps (measured or estimated) states to control actions for a specific choice of program and code generation parameters; (1d) is a stochastic disturbance that evolves according some probability distribution $P_{w_t}(x_t, u_t)$ that is conditionally independent of previous disturbance realizations given the current states and actions; (1e) represents the ith performance metric J_i defined in terms of the closed-loop system evolution given local stage cost $\ell_{t,i}$ and terminal cost $\ell_{T,i}$ functions over a finite time horizon T; (1f) denotes a hardware resource utilization constraint represented by a binary function $g : \Theta \times \Gamma \to \{0, 1\}$ that indicates if the high-level control program can be compiled and executed on the available hardware (i.e., $+1$) or not (i.e., 0); and (1g) represents the user-defined search space of control programs and code generation strategies. The expectation $\mathbb{E}\{\cdot\}$ in (1e) is taken with respect to the stochastic disturbance sequence of the closed-loop system $\{w_0, \ldots, w_{T-1}\}$.

The MOO problem (1) represents a very general framework for CoC design. In fact, we can interpret virtually all end-to-end control design procedures as a special case to (1). However, approximations are necessary in practice due to the

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

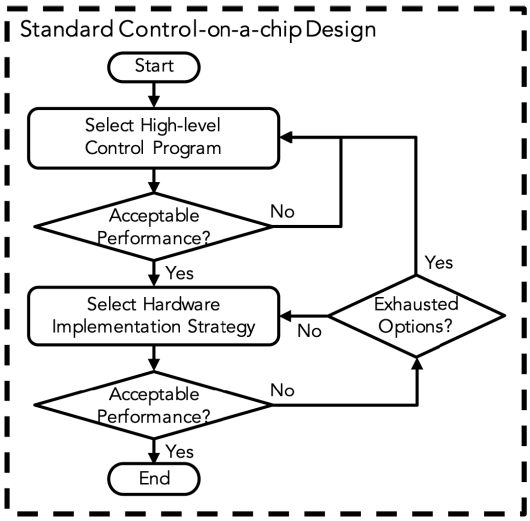

Fig. 1. Flow diagram of the standard CoC design process. First, a high-level representation of the control policy is selected and evaluated using approximate models or limited closed-loop data. Then, a code generation strategy is selected and evaluated based on its ability to be successfully implemented while matching the performance of the high-level program. In general, several iterations may be needed at each stage of the design process until an acceptable option is found. If any stage fails, then one must return to a previous stage to repeat the process.

intractability of (1), which stems from two main challenges. First, the program parameter space Θ is abstract. To ensure that it has sufficiently large representation power, the control policy will typically be embedded in some high-dimensional space $\Theta \subset \mathbb{R}^D$, where D can be very large. Furthermore, this space may involve discrete and continuous variables that are needed to represent logical relationships between different sets of variables. Second, we often do not have exact knowledge of the system dynamics $f(\cdot)$, disturbance distribution $P_{w_t}(x_t, u_t)$, and the hardware utilization constraint function $g(\cdot)$, which prevents the application of traditional MOO methods that require equation-oriented forms for all objective and constraint functions.

Standard control design approaches often proceed in the following steps that are illustrated in Fig. 1: 1) restrict the control policy representation by constraining Θ to describe a narrow set of policies in a low-dimensional space; 2) independently search for control program parameters θ that approximately minimize the closed-loop performance metrics either using approximate models or (limited) closed-loop data; and 3) search for code generation parameters γ that enable the desired θ to be executed on the available hardware. If step 3 fails, then one must go back and repeat the process again for a narrower set of more computationally tractable policies. For example, complex control policy formulations as in optimization-based control [e.g., as described in Section III-C, see (5)] may require specific code libraries and/or routines that are not easily implemented on low-resource hardware due to memory restrictions or reduced accuracy due to quantized numeric representation. Furthermore, the emergence of advanced hardware (e.g., GPUs and FPGAs) to speed up computations requires additional low-level code generation that requires specialized knowledge of the device architecture. Separate design of θ and γ misses out on important interactions between high-level program representation and code generation. In particular, the hardware utilization constraint function $g(\cdot)$ represents whether or not the policy can be embedded into the hardware or can satisfy timing constraints. Examples of parameters that influence this constraint include numeric representation, a number of operations, parallelization options, and so on. Without co-design, there is a much greater chance that CoC policies cannot be implemented as knowledge of how the algorithmic requirements are translated to the physical wires in hardware is not generally accessible.

The main goal of this work is the development of an iterative learning-based strategy capable of systematically tackling the *hardware–software co-design* problem (1). The core structure of our proposed strategy, which relies on deep learning to simplify the policy and hardware utilization constraint, is presented in Section III. In Section IV, we then describe an efficient multiobjective black-box optimization strategy that takes advantage of this structure by searching over a reduced set of parameters.

III. Universal Policy Representation for Feasible CoC Systems

The lack of an easy-to-search program space Θ and known structure for the feasibility constraint g greatly complicates the traditional CoC design procedure (Fig. 1). In this section, we show how both of these problems can be addressed by working with DNN policies such that $\theta = \{\theta_W, \theta_A\}$ can be separated into continuous weight and bias parameters θ_W and architecture parameters θ_A that can be discrete. This not only helps us simplify the learning process for g but also allows us to take advantage of prior knowledge to "train" θ_W such that we only consider a small subset of parameters when optimizing closed-loop performance.

A. Deep Learning for Control Policy Representations

Deep learning is a generalized term for computational structures/graphs characterized by multiple "layers." Through multiple layers, deep learning transforms an input representation to abstract representations until the ultimate output is learned [37]. It is exactly many layers that allow deep learning to extract (on its own) features from raw data that are relevant to learning control policies [29], [38].

The key advantage of DNN policies is the surprisingly robust ability to train such large structures using (stochastic) gradient descent style methods. For simplicity of presentation, consider a fully connected feedforward DNN control policy $\pi_{\text{dnn}}(x; \theta)$ with L hidden layers and H nodes per layer, which can be mathematically defined as follows:

$$\pi_{\text{dnn}}(x; \theta) = \alpha_{L+1} \circ \beta_L \circ \alpha_L \circ \cdots \beta_1 \circ \alpha_1(x) \tag{2}$$

where $\alpha_1(x) = W_1 x + b_1$ is an affine transformation of the input, $\alpha_l(z_{l-1}) = W_l z_{l-1} + b_l$ are affine transformations of the hidden layers for all $l \in \{2, \ldots, L+1\}$, $\beta_l(z)$ are nonlinear activation functions (e.g., $\beta_l(z) = \max\{z, 0\}$ for ReLU activation functions) for all $l \in \{1, \ldots, L\}$, and $\theta_W = \{W_1, b_1, \ldots, W_{L+1}, b_{L+1}\}$ denotes the collection of

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

weights and biases that parameterize the network for a fixed architecture θ_A (e.g., type of activation function, L, and H). Due to their continuous representation, θ_W can be trained by minimizing a loss function that captures how well the DNN performs on a given task. This process is known to work well in practice under the assumption that the gradient of the loss function with respect to θ_W can be efficiently computed via backpropagation [39]. This is not necessarily the case when one attempts to use π_{dnn} in (1) unless a differentiable structure for the dynamics and cost functions is known.

Remark 1: Note that the DNN policy defined in (2) is just one choice of architectural representation of artificial neural networks. In fact, any deep learning architecture can be used to approximate (2). For example, if the state or any other exogenous signals involved image data, we could exploit a convolutional neural network structure that is designed to specifically exploit the regularity of image pixel patterns.

B. Learning Feasible Space of CoC Policies

Given that our ideal policy is represented by a DNN, in addition to the program (i.e., software) (2), CoC policies also require specification of the embedded version of the program that can be run on the actual hardware. Although this difference is not practically important in the absence of resource limitations, it is very important in cases where there are constraints on the number of real-time computations and/or resource (memory or power) utilization [40]. A useful property of DNNs is that the resource utilization is the same for all θ_W given a fixed architecture θ_A. To see this, we can compute the number of operations N_{op} for a dense DNN of the form (2) as

$$N_{\text{op}} = (n_{\text{in}} + 1)H + H(H + 1)L + (H + 1)n_{\text{out}} \tag{3}$$

where n_{in} and n_{out} are the number of inputs and outputs, respectively. Thus, by simply changing the architecture of the DNN (e.g., reducing the number of nodes or layers), we can lower the evaluation cost on hardware.

Nonetheless, we cannot use N_{op} to directly characterize the feasible set of CoC policies, i.e., $\mathcal{F} = \{(\theta, \gamma) \in \Theta \times \Gamma : g(\theta, \gamma) = 1\}$, since this set will depend on how the operations written in a mid- or high-level programming language (e.g., MATLAB, Python, and C) get translated to low-level machine code (e.g., assembly language and binary), compiled, and then packaged. Automatic code generation tools aim to provide a streamlined means of performing such tasks by abstracting the laborious translation process [41], [42], [43]. As such, code generation serves as a bridge between human-interpretable code and machine-interpretable instructions. In this work, we treat code generation as a black-box function

$$\pi(\cdot; \theta, \gamma) = \mathcal{CG}(\pi_{\text{dnn}}(\cdot; \theta), \gamma) \tag{4}$$

that takes as input a DNN policy and some parameters related to the translation process γ and returns a machine-interpretable policy. Since changing θ_W will not fundamentally change the structure of the returned CoC policy π, the function g will be independent of θ_W and, thus, we only need to learn an approximation of $g(\theta_A, \gamma)$.

One way to learn an approximation $\tilde{g} \approx g$ is to run the code generation process $\mathcal{CG}$ for a randomly generated DNN for several values of $(\theta_A, \gamma) \in \Theta_A \times \Gamma$, which is expected to be a much lower dimensional space than Θ_W. The outcome can be recorded as either a successful compilation 1 or failed compilation 0. These labeled data can then be used to train a binary classifier that is capable of predicting whether a new choice of architecture and code generation parameters is feasible or not. To this end, any classifier type can be used, for example, support vector machines and DNN with a sigmoid activation function at the output layer. We highlight that the major advantage of this approach is that $\tilde{g}$ can be trained *independently* of the quality of the CoC policy. Not only can this process be done fully offline, but also it does not require any system data to be generated—all that is required is access to the hardware and code generation process.

C. Accelerated Training of Hardware-Feasible DNN Policies by Imitating Physics-Informed Expert Policies

In Section III-B, we presented an efficient way to verify whether a CoC policy will be feasible. Yet, the original MOO problem (1) can still be computationally intractable since θ_W remains a high-dimensional space with possibly thousands or more independent parameters. The question we address here is how the search over this space can be efficiently performed without sacrificing the achieved closed-loop performance. To this end, we rely on a class of control policies that are implicitly defined in terms of a set of interpretable set of equations [23]. Specifically, we look to use policies defined by an optimization problem

$$\pi_{\text{opt}}(x; \lambda) = \arg\min_{u} \; V(x, u; \lambda) \tag{5a}$$

$$\text{s.t. } h_i(x, u; \lambda) \le 0, \quad i = 1, \ldots, k \tag{5b}$$

$$g_i(x, u, \lambda) = 0, \quad i = 1, \ldots, r \tag{5c}$$

where $V : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ is the objective function; $h_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ are the inequality constraints for all $i = 1, \ldots, k$; $g_i : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \times \mathbb{R}^{n_\lambda} \to \mathbb{R}$ are the equality constraints for all $i = 1, \ldots, r$; and $\lambda \in \Lambda \subset \mathbb{R}^{n_\lambda}$ are tunable policy parameters. As discussed in [23], this representation captures a large set of policies, including approximate dynamic programming and MPC. The key idea behind (5) is that (5a) describes some type of value or reward function, (5b) represents critical state and/or input constraints, and (5c) represents a (possibly physics-based) model of the system. A significant advantage of the structure (5) is that it provides a natural way for users to incorporate prior knowledge about the system, when available, by properly selecting or constraining the functions $V, h_1, \ldots, h_k$, and $g_1, \ldots, g_r$.

The central notion is that λ can be much lower dimensional than θ_W such that we can derive an explicit value for the DNN parameters that depend on λ by minimizing the error between the DNN policy π_{dnn} and the "physics-informed" expert policy π_{opt}

$$\theta_W^\star(\lambda, \theta_A) = \arg\min_{\theta_W} \frac{1}{n_s} \sum_{i=1}^{n_s} \left\| \pi_{\text{opt}}\left(x^{(i)}; \lambda\right) - \pi_{\text{dnn}}\left(x^{(i)}; \theta_W, \theta_A\right) \right\|^2 \tag{6}$$

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

where $\{(x^{(i)}, \pi_{\text{opt}}(x^{(i)}; \lambda))\}_{i=1}^{n_s}$ represent a set of n_s state–action pairs acquired by solving the optimization problem (5) offline for specific state values and fixed λ values. This dataset can be generated in a variety of ways, including random sampling in the state space or using closed-loop "rollouts" from likely initial conditions [38]. It is important to note that special care must be taken in generating the dataset to train the DNN as the approximation of (5) may reduce the robustness properties ensured by implementing it directly depending on the quality of the overall training process. In accordance with the universal approximation theorem [44], there exists a DNN that matches (5) exactly under some relatively mild continuity assumptions. As such, given a sufficiently large architecture and enough training data, we can ensure that $\pi_{\text{dnn}}(\cdot; \theta_W^\star(\lambda, \theta_A), \theta_A) \to \pi_{\text{opt}}(\cdot; \lambda)$ for some θ_A.

Notice that the solution to (6) will depend on both the expert policy parameters λ and the DNN architecture hyperparameters θ_A. Therefore, the proposed CoC policy has the following unique structure:

$$\pi_{\text{CoC}}(\cdot; \lambda, \theta_A, \gamma) = \mathcal{CG}\big(\pi_{\text{dnn}}\big(\cdot; \theta_W^\star(\lambda, \theta_A), \theta_A\big), \gamma\big) \tag{7}$$

which depends on three sets of parameters, mainly λ, θ_A, and γ that all appear in different components of the CoC framework. An illustration of the proposed CoC design process is shown in Fig. 2, which is based on selecting $(\lambda, \theta_A, \text{ and } \gamma) \in \Lambda \times \Theta_A \times \Gamma$ to optimize closed-loop performance metrics of interest. However, we do not have a closed-form expression for how performance metrics depend on $(\lambda, \theta_A, \text{ and } \gamma)$. This is further compounded by the cost of collecting closed-loop performance data since it requires: 1) training a DNN policy; 2) executing a code generation process to run the policy on embedded hardware; and 3) running hardware-in-the-loop closed-loop experiments to collect performance data. Next, we present an efficient procedure for searching over this joint parameter space.

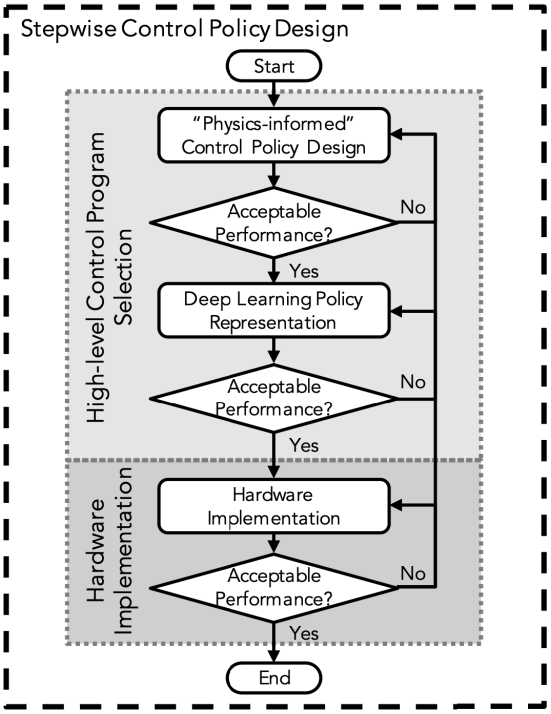

Fig. 2. Flow diagram of the proposed CoC design process. As in Fig. 1, CoC design is subdivided into two categories related to software (high-level control program selection in light gray) and hardware implementation (in dark gray). Our proposed workflow for CoC design is subdivided as follows. Within the high-level control program selection, the first step is to select and evaluate a "physics-informed" control design. In the next step, a deep learning-based policy is created in pursuit of hardware compatibility. The final step involves the hardware implementation and final evaluation. Note that the "Exhausted Options" decision-marker is not present in this figure for simplicity but still exists as part of the design process. We define this design process as a framework for CoC design that can be used to search over the joint software and hardware parameter space.

IV. Proposed Multiobjective Optimization Framework for Hardware–Software Co-Design

The system dynamics and CoC policy together form a stochastic process due to the initial condition x_0 and disturbances $\{w_t\}_{t\geq 0}$ that are random variables

$$x_{t+1} = f(t, x_t, u_t, w_t), \quad t = 0, 1, \ldots \tag{8a}$$

$$u_t = \pi_{\text{CoC}}(x_t; \xi) \tag{8b}$$

where $\xi = (\lambda, \theta_A, \gamma)$ is the concatenation of all software and hardware parameters that define the policy. A specific choice of ξ can be judged according to the set of M expected closed-loop performance metrics $J_i(\xi)$, $i = 1, \ldots, M$, defined in (1) with the state and input sequences generated by (8).

Since J_i are defined as expectations over closed-loop trajectories, they only depend on ξ that is of much lower dimensional than the original θ space, as discussed above. Thus, we now pose (1) as a more manageable MOO problem

$$\min_{\xi \in \Xi} \{J_1(\xi), \ldots, J_M(\xi)\} \tag{9}$$

where $\Xi = \{(\lambda, \theta_A, \gamma) \in \Lambda \times \Theta_A \times \Gamma : \tilde{g}(\theta_A, \gamma) = 1\}$ is the space of CoC parameters that can be compiled on the available hardware. Since the functions $\{J_i\}_{i=1}^M$ are black box in nature, we must resort to derivative-free optimization (DFO) methods to approximately solve (9) in practice. In particular, the DFO method must be able to handle noisy, expensive evaluations of $\{J_i\}_{i=1}^M$. The evaluations are expensive due to the need to collect hardware-in-the-loop data, as discussed previously. These evaluations will also be subject to noise due to the expectation operator that defines J_i. In practice, we can approximate this expectation using a random sampling technique (e.g., Monte Carlo sampling) such that $y_i = J_i(\xi) + \varepsilon_i$, where ε_i is the effective measurement noise in the i-th performance function. Assuming that K independent random samples are used to approximate the performance functions, then it is known that ε_i approaches a zero-mean Gaussian random variable whose variance decreases at a rate of $1/K$ by the central limit theorem [45].

We briefly highlight the fact that the only major assumption made in (9) is that we can generate independent noisy measurements of the closed-loop performance functions. We do not require any specific knowledge of the dynamics, or uncertainty distribution, which makes the proposed hardware–software co-design approach broadly applicable. However, the more knowledge that we can exploit in the specification of the expert policy (5), the better the choice of the $\xi \in \Xi$ space, which can simplify the process of solving (9).

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

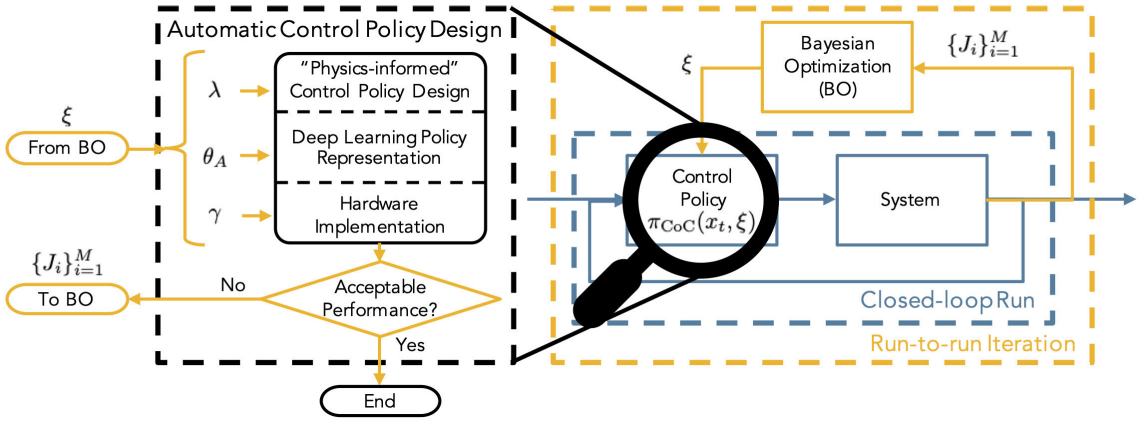

Fig. 3. Diagram of the data-driven optimization framework. The optimization framework consists in: 1) an inner learning procedure that represents a templated workflow to design a single CoC policy (black dashed box) and 2) an outer optimization stage that suggests new CoC designs via closed-loop evaluations (yellow dashed box). The inner learning procedure 1) is similar to Fig. 2, but rather than iteratively optimizing between steps as in Fig. 2, the outer optimization 2) allows us to select parameters from each step (λ, θ_A, and γ) concurrently.

A. BO Approach to Co-Design

Since $\{J_i\}_{i=1}^M$ are noisy, expensive functions defined over a relatively low-dimensional space $\xi \in \Xi$, BO is a natural choice of DFO framework for (9) since it is specifically designed for such cases. Furthermore, BO has been shown to surpass state-of-the-art performance in real-world controller tuning applications with a variety of policy types [23]. BO falls under the paradigm of *active learning*, meaning that it translates the optimization task into an iterative learning task. There are two major components in BO. First, we must construct a *probabilistic surrogate model*, typically a Gaussian process (GP) [35], to provide a posterior distribution $\mathbb{P}\{\boldsymbol{J}|\mathcal{D}_n\}$ over the unknown true vector-valued function values $\boldsymbol{J}(\xi) = (J_1(\xi), \ldots, J_M(\xi))$ given a prior dataset $\mathcal{D}_n = \{(\xi_i, \boldsymbol{y}_i)\}_{i=1}^n$. Second, we must define an *acquisition function* $\alpha_n : \Xi \to \mathbb{R}$ that uses the surrogate model to assign a utility value to the future candidate points at which we can evaluate the true function. Thus, for a well-designed α_n, we would like to preferentially sample at a point that produces the highest possible value. The active learning process is then defined by

$$\xi_{n+1} = \arg\max_{\xi \in \Xi} \alpha_n(\xi). \tag{10}$$

Since the surrogate approximation of $\boldsymbol{J}$ is expected to be much cheaper than the true function, we can (approximately) solve (10) using established optimization algorithms.

Remark 2: While GPs are the standard surrogate model-of-choice for BO, they are known to scale poorly with the number of data points D, requiring $O(D^3)$ floating-point operations for exact inference, and a higher number of data points may be required to obtain representative models for higher dimensional problems. However, there are recent advances that reduce the computational cost at the cost of accuracy (e.g., [46], [47]). Additional ways to address large data problems, including using a different type of surrogate model, are an active area of open research [48], [49].

Since we are interested in MOO, we do not have a single best solution and instead would like to provide the control practitioner with an estimate of the set of *Pareto optimal* solutions. A point $\xi \in \Xi$ is considered to be Pareto optimal if improvement in one objective means deteriorating one or more of the others. The so-called Pareto frontier is the set of Pareto optimal points, which is mathematically defined as

$$\mathcal{P}^\star = \left\{\boldsymbol{J}(\xi) : \nexists \xi' \in \Xi, \text{s.t. } \boldsymbol{J}(\xi') \succ \boldsymbol{J}(\xi)\right\} \tag{11}$$

where $\boldsymbol{J}(\xi') \succ \boldsymbol{J}(\xi)$ implies that the point ξ' dominates the point ξ, which occurs if $J_i(\xi') \geq J_i(\xi)$ for all $i = 1, \ldots, M$. To derive α_n, we would like to select points that grow our understanding of $\mathcal{P}^\star$. Following previous work [30], [50], [51], we use the expected hypervolume improvement (EHVI) acquisition function defined as follows:

$$\alpha_n(\xi) = \mathbb{E}\{\text{HV}(\mathcal{P} \cup \{\boldsymbol{J}(\xi)\}, \boldsymbol{r}) - \text{HV}(\mathcal{P}, \boldsymbol{r})\} \tag{12}$$

where $\text{HV}(\mathcal{P}, \boldsymbol{r})$ denotes the hypervolume of a finite approximate Pareto set $\mathcal{P}$ and a reference point $\boldsymbol{r} \in \mathbb{R}^M$ that bounds $\mathcal{P}$ from below. The HV can be computed exactly as the M-dimensional Lebesgue measure

$$\text{HV}(\mathcal{P}, \boldsymbol{r}) = \lambda_M\left(\bigcup_{i=1}^q [\boldsymbol{r}, \boldsymbol{y}_i]\right) \tag{13}$$

where $\mathcal{P} = \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(q)}\}$ is composed of a finite set of q points. EHVI is notoriously difficult to optimize since it has a relatively higher computational cost compared to standard single-objective BO acquisition functions (such as expected improvement) when evaluated using box decomposition. However, as shown in [50], one can more efficiently compute EHVI and its gradients exactly (up to a Monte Carlo integration error) using the inclusion–exclusion principle [52], making the solution of (10) tractable.

An illustrative summary of the complete hardware–software co-design framework is provided in Fig. 3.

B. Kernel Selection for Multiobjective Controller Tuning

The choice of the covariance (or kernel) function in the GP model for $\boldsymbol{J}$ is a critical parameter in the proposed MOBO approach. A particular challenge is the fact that ξ may consist of *ordinal* and *categorical* variables. Ordinal variables are

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

those with some type of natural ordering such as continuous (e.g., weight parameters in the control policy) and integer variables (e.g., the number of nodes/layers in the DNN policy). Categorical variables, on the other hand, are best described as a collection of unordered categories such as the choice of parallelization option in the code generation tool. The standard way for dealing with categorical variables in GP models is to apply one-hot encoding (i.e., converts a c-category variable into c new binary variables). The main challenge with one-hot encoding is that it can lead to a large increase in the dimensionality of the search space as well as complicate the acquisition optimization process. As such, we pursue a mixed kernel function approach [36] that combines separate kernels for the ordinal and categorical variables. For every element of $\boldsymbol{J}$, we focus on independent kernels of the following form:

$$k(\xi,\xi') = k^1_{\text{cat}}(\xi_{\text{cat}},\xi'_{\text{cat}})k^1_{\text{ord}}(\xi_{\text{ord}},\xi'_{\text{ord}}) + k^2_{\text{cat}}(\xi_{\text{cat}},\xi'_{\text{cat}}) + k^2_{\text{ord}}(\xi_{\text{ord}},\xi'_{\text{ord}}) \tag{14}$$

where ξ_{cat} and ξ_{ord} denote the categorical and ordinal components of ξ, respectively; and k^1_{cat}, k^2_{cat}, k^1_{ord}, and k^2_{ord} are kernels associated with the categorical and ordinal variables, each with their own set of hyperparameters. The k^1_{cat} and k^1_{ord} kernels are associated with a product in (14), so we can capture the joint impact of both types of variables. The k^2_{cat} and k^2_{ord} kernels are associated with a summation in (14), so we focus on independent impacts of each type of variable.

Based on the idea of Hamming distances, the categorical kernel is selected as follows:

$$k^i_{\text{cat}}(x,y) = \nu_i \exp(-d(x,y)/l_i), \quad i \in \{1,2\} \tag{15}$$

where $d(x,y)$ is meant to represent the distance between categories (equal to 0 if $x = y$ and 1 otherwise) and ν_i is a variance hyperparameter related to the magnitude of the function and l_i is a lengthscale hyperparameter related to how fast the function can vary with distance. For the ordinal variables, we focused on a Matérn-5/2 kernel that has similar variance and lengthscale hyperparameters, though this choice could easily be replaced with any other established kernel (see [35] for details on kernel choices and properties).

V. Hardware Considerations With Deep Learning: An Illustrative Example

As a first look into the complexity of the embedded control problem, we examine the benefits offered by using deep learning as a bridge between hardware and software. A major consideration in embedded control is whether or not the proposed control policy can be implemented on hardware, which is related to the hardware constraint (1f), and can be a factor of the online computational complexity of the control policy. This section examines the online evaluation of various control policies to illustrate the need for hardware considerations during the software policy development.

Consider a system of masses attached by springs as given in [53]. In general, for a system of m masses, the system can be described by a linear state-space model with the m positions and m velocities of each mass as the states and applied force to $m-1$ masses as the inputs. Due to hardware considerations,[1] we use a simplified two-mass system. A first-order hold discrete-time model with a sampling time of 0.5 s is derived from the continuous-time dynamics such that the plant is of the form

$$x_{t+1} = Ax_t + Bu_t + Gw_t \tag{16}$$

for fixed (A, B, G) matrices given in [53]. We assume that the disturbances w_t are independent uniform random variables acting on each mass with each element between -0.5 and 0.5. We can then derive a nominal MPC policy whose goal is to keep the system at rest starting from an initial condition of $x_0 = 0$ by solving the minimization problem

$$\begin{aligned} \min_{\{u_{k|t}\}_{k=0}^{N-1}} & \sum_{k=0}^{N-1} x_{k|t}^\top Q x_{k|t} + u_{k|t}^\top R u_{k|t} \\ \text{s.t. } & x_{k+1|t} = Ax_{k|t} + Bu_{k|t}, \quad k = 0,\dots,N-1 \\ & x_{\min} \le x_{k|t} \le x_{\max}, \quad k = 1,\dots,N \\ & u_{\min} \le u_{k|t} \le x_{\max}, \quad k = 0,\dots,N-1 \end{aligned} \tag{17}$$

where $Q = \text{diag}(1,1,1,1)$ and $R = 1$ are the state and input weight matrices, respectively, and the state and input bounds are given by $x_{\min} = [-4,-4,-4,-4]^\top$, $x_{\max} = [4,4,4,4]^\top$, $u_{\min} = -0.5$, and $u_{\max} = 0.5$.

We evaluate the embeddability of three control policies: an implicit MPC, an EMPC, and a DNN approximation to MPC. Implicit MPC refers to the online solution to optimization problem (17). We use CasADi to formulate the problem as a quadratic program and solve using QRQP [54]. EMPC subdivides the state-space into polytopic regions with precomputed controller gains. EMPC can typically be readily embedded on hardware since the control law is reduced to a lookup table [55]. The optimization problem is created with the MPT-3 toolbox [56] by using the `MPCController()` function, which is the equivalent to the implicit MPC created using CasADi. An explicit control policy is then created by calling the `toExplicit()` function, which solves a multiparametric programming problem to determine the piecewise affine (PWA) function that replaces the online optimization problem. The evaluation of the explicit control policy is performed by conducting sequential search on the returned set of gains for the PWA representation. Note that when the EMPC is exported to C-code, the search is conducted via more efficient binary search trees [56]. Finally, the DNN-based approximate MPC is trained using data generated offline by solving the implicit MPC law [see (17)]. The DNN is trained following the procedure described in Section III-C using $n_s = 5000$ samples with MATLAB's `feedforwardnet` function. Note that the choice of n_s and the quality of the training data must be done with respect to the desired accuracy and robustness, where the choice of $n_s = 5000$ was done empirically by performing some initial training, validation, and testing results on randomly selected architectures.

Each method was evaluated across 1000 replicate simulations with 100 time steps each resulting in 100 000 total

[1]The exact system described in [53] (12-state, 3-input) is computationally challenging for standard explicit MPC (EMPC) tools, taking more than five days to generate the solution offline and using more than 30 000 lookup values.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
COMPARISON OF CONTROL POLICIES IN THE ILLUSTRATIVE EXAMPLE ON GENERAL-PURPOSE CPU: MPC, STANDARD EMPC, AND NEURAL NETWORK APPROXIMATION OF MPC (DNN)

	Stage Cost	Computation Time (ms)			Memory (kB)	
	Average	Average	Max	Min	File Size	Max Heap
MPC	4.77 ± 5.74	0.55 ± 0.11	1.91	0.37	242.5	5.0
EMPC	4.81 ± 5.92	1.40 ± 0.13	2.76	1.32	5600.0	6.8
DNN	4.76 ± 5.67	0.01 ± 0.01	0.38	0.01	18.0	5.0

time steps. Disturbance realizations were consistent for each controller but varied in each replicate simulation. To mimic embedded implementation, each control policy was converted to C-code and compiled into `MEX` functions. Each of CasADi, MPT-3, and MATLAB has a routine to convert the m-code into C-code. Once the C-code is generated, each program is compiled into a `MEX` function. A `MEX` function is the MATLAB function that calls a C program, acting as an interface between a high-level programming language (i.e., MATLAB) and a low-level one (i.e., C). Settings and the procedures used in the compilation process are detailed in the Appendix.[2]

Table I compares three metrics of interest for each control policy: closed-loop performance, computational time, and memory requirements. The closed-loop performance is given as the average stage cost computed over the 100 000 steps of the system. In addition, Fig. 4 depicts the distributions of the stage cost for each controller. The computational time is given as the average time taken to compute an input using a compiled C-program (via `MEX` function) on a general-purpose laptop CPU (2.3-GHz 8-Core Intel i7 processor).[3] The memory requirements are given in two forms: one related to the general storage of the program (executable file size) and one related to the random access memory (RAM) required during computation (maximum heap utilization). The closed-loop performance between the three controllers is similar, if not the same. However, when considering hardware constraints/utilization, standard EMPC takes the longest computation time and greatest memory consumption. In general, standard EMPC is ill-suited for this problem since even the implicit solution can outperform EMPC. Regardless, the DNN can offer an order-of-magnitude improvement in computation speed and memory utilization compared to the MPC. The gap between MPC implementations and DNN implementations can be significantly widened in specialized contexts, e.g., in cold plasma bioprocessing/medicine, where the dynamics are nonlinear and the control context requires safety considerations.

[2]All code and additional documentation (including those for the results and discussion of the case study in Section VI) may be found at https://github.com/Mesbah-Lab-UCB/HW-SW_CoDesign4CoC.

[3]On a less powerful device and/or on specialized hardware, the results in Table I would show even larger differences.

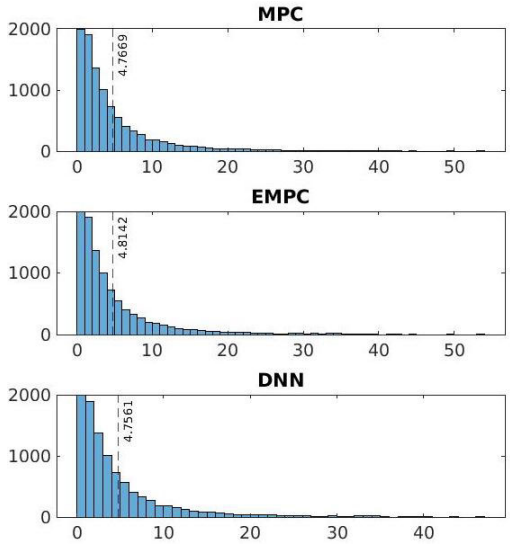

Fig. 4. Distributions of the stage cost for three control policies used in the illustrative example: implicit MPC, standard EMPC, and DNN approximation to MPC (DNN).

VI. APPLICATION TO ROBUST PREDICTIVE CONTROL OF APPJS

To illustrate the complete proposed design framework illustrated in Fig. 3, we investigate the CoC design for cold atmospheric plasma (CAP) devices. CAPs have recently found promising use in a variety of applications, including (bio)materials processing [57] and plasma medicine [58], [59], [60]. CAPs, a low-temperature (partially) ionized gas, can be generated by applying electric fields to a noble gas, typically argon or helium. The synergistic effects of CAPs, including the generation of reactive chemical species and ions, ultraviolet radiation, low-level electric fields, and thermal effects, are posited to induce therapeutic and practical outcomes [61], [62]. CAP devices, such as APPJs [63], can facilitate direct CAP treatments by providing a portable, point-of-use solution to deliver plasma effects in a directed manner. However, APPJs pose unique challenges in control and rely on cutting-edge control formulations [38], [64], most of which have no unified embedded techniques. This section will extensively cover the description of the APPJ system and demonstrations of a CoC design study.

A. Description of the APPJ Testbed

We use a kilohertz-excited APPJ in helium that consists of a copper ring electrode wrapped around a quartz tube, which serves as a dielectric barrier and the gas flow channel. A schematic of the APPJ is shown in Fig. 5. Helium gas flows through the tube, and plasma ignition is achieved by applying a high-frequency, alternating current (ac) voltage to the copper electrode. The generated plasma is directed out of the tube onto a grounded, glass-covered metal plate at a distance of 4 mm below the tip of the tube. In this testbed, the applied power P in watts and the helium flow rate q in standard liters per minute (s.l.m.) are the manipulated inputs. The maximum surface temperature T in degrees

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

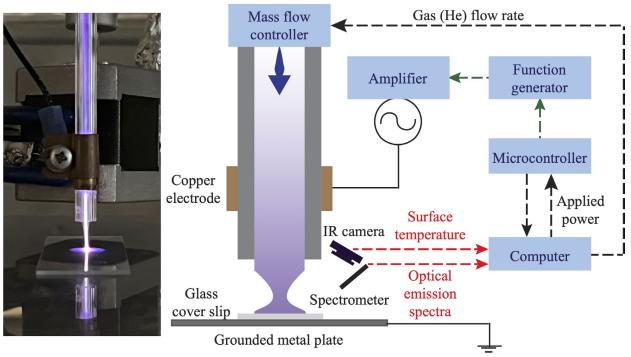

Fig. 5. Close-up image (left) and schematic (right) of the kilohertz-excited APPJ in helium (He). The manipulated inputs are denoted along the black dotted arrows, and the controlled outputs are denoted in red.

Celsius and the total optical intensity of the plasma I in arbitrary units (arb. units) at the plasma-surface incident point are the measured outputs made available every 0.5 s. In the experimental testbed, the applied power is implemented using an embedded proportional–integral (PI) controller on a microcontroller (Arduino UNO) that manipulates the applied voltage. Furthermore, the applied voltage signal is created by generating a sinusoidal waveform at a specified frequency using a function generator (integrated circuit, XR-2602CP). This signal is amplified using an amplifier (TREK 10/40A-HS) before being sent to the copper electrode. Surface temperature is measured through a radiometric infrared thermal camera (Lepton FLIR 3) and optical plasma intensity is measured via a fiber optic cable connected to an optical emission spectrometer (Ocean Optics USB2000+). Additional information on this testbed may be found in [65]. Data acquisition is implemented and managed via USB connection to any standard CPU using Python. DNN-based CoC policies were implemented on an FPGA (the programmable logic side of a Zybo Z7, XC7Z020-1CLG400C). The programming files for the FPGA were generated automatically using MathWorks HDL Coder (included with MATLAB R2021a) and Xilinx Vivado 2020.1. FPGA-in-the-loop simulations and experiments were facilitated by MATLAB on a standard laptop CPU, rather than the Zybo Z7's onboard processor.[4]

B. Modeling for Control of APPJs

A critical challenge in the development of model-based optimal control policies lies in the modeling of the CAP and its interactions with the target surface. CAPs are notoriously difficult to model since they exhibit nonlinear dynamics that are distributed over multiple length and time scales. Modeling difficulty is further exacerbated by the intrinsic variability in the plasma and sensitivity of the APPJ to exogenous disturbances. Moreover, the use of theoretical models [66], [67], [68] is ill-suited for real-time control of plasma effects that occur on the millisecond-to-second time scale. Instead, a common solution is to resort to data-driven modeling of the APPJ [69], [70].

[4]Note that the communication time between devices can play a role in effective hardware implementation. Further investigation into using a system-on-chip architecture provided by the Zybo Z7 is left for future work.

Here, we identify a linear, time-invariant model using the `n4sid` function in MATLAB using input–output data of the APPJ. Input–output data were gathered by performing multiple-step tests in the inputs $u = [P, q]^\top$ and recording the outputs $y = [T, aI]^\top$, where a is a scaling factor to scale the total intensity to the same order of magnitude as surface temperature. Furthermore, the data were centered around nominal operating conditions $[P^s, q^s]^\top = [1.5\ \text{W},\ 3.5\ \text{s.l.m.}]^\top$ and $[T^s, I^s]^\top = [33.3\ ^\circ\text{C},\ 7.7\ \text{intensity units}]^\top$, where superscript s denotes the nominal condition. The model follows the discrete-time state-space form:

$$x_{t+1} = Ax_t + Bu_t \tag{18a}$$

$$y_t = Cx_t + Du_t \tag{18b}$$

where $t \geq 0$ is the discrete-time step; $x \in \mathbb{R}^{n_x}$ is the vector of states; $u \in \mathbb{R}^{n_u}$ is the vector of manipulated inputs; $y \in \mathbb{R}^{n_y}$ is the vector of measured outputs; and A, B, C, and D are the state-space matrices identified using subspace identification [71]. The state-space model is defined in terms of deviation variables around the nominal operating condition, i.e., $y = [(T - T^s), (aI - I^s)]^\top$ and $u = [(P - P^s), (q - q^s)]^\top$. In this case, we assume an observable canonical form of (18), where $C = \boldsymbol{I}$ and $D = \boldsymbol{0}$ (see Appendix A for a detailed description of model matrices). In the closed-loop simulation studies, the true system model is treated as having a white noise term added to (18)

$$f(t, x_t, u_t, w_t) = x_{t+1} = Ax_t + Bu_t + w_t \tag{19}$$

where w_t is generated from a uniform distribution with all elements bounded in $[-1, 1]$.

Plasma treatment not only depends on the current state of the plasma itself but also on quantification of the delivered plasma effects to a surface. While quantification of plasma effects is generally cumbersome and application dependent [72], we take inspiration from hypothermia treatments to quantify the delivery of a desired thermal effect (also known as a thermal dose) [73]. A thermal dose metric is quantified in terms of cumulative equivalent minutes (CEMs), which describes the accumulation of thermal effects on a target with respect to a reference temperature. The CEM is described by

$$\text{CEM}_{t+1} = \text{CEM}_t + 0.5^{(T_{\text{ref}} - T_t)}\delta t \tag{20}$$

where $T_{\text{ref}} = 43\ ^\circ\text{C}$ is the reference temperature and δt is the sampling time. This definition of the thermal dose is cumulative, in that plasma effects delivered cannot be removed, and nonlinear due to the exponential dependence on temperature. Our control objective is to deliver a desired "dose" of thermal effects given by a target CEM value.

C. Scenario-Based MPC

Since we have no way to remove thermal effects once delivered, we need to be cautious in the face of uncertainty. Therefore, we select scenario-based MPC [74], or sMPC for short, to provide a controller that is robust to uncertainty, especially in the presence of safety-critical constraints. Specifically, sMPC assumes that the system uncertainty can take on a finite number of s scenarios at every time step.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

Whenever the uncertainty is time varying in the sense that it can take on new values at every time step, the system evolution can be represented by a *scenario tree* of $S = s^N$ unique combinations of uncertainty values where N denotes the prediction horizon [75]. The sMPC policy then solves the following minimization problem at every time step:

$$\min_{u_{k,j|t}} \sum_{j=1}^{S} \omega_j \left[\sum_{k=0}^{N-1} L\left(x_{k,j|t}, u_{k,j|t}\right) + L_f\left(x_{N,j|t}\right) \right] \tag{21a}$$

$$\text{s.t. } x_{k+1,j|t} = Ax_{k,j|t} + Bu_{k,j|t} + w_{k,j|t}$$

$$\left(x_{k,j|t}, u_{k,j|t}\right) \in \mathcal{X} \times \mathcal{U} \tag{21b}$$

$$\sum_{j=1}^{S} \tilde{E}_j U_{j|t} = 0 \tag{21c}$$

$$x_{0,j|t} = x_t$$

$$\forall k \in \{0, \ldots, N-1\} \quad \forall j \in \{1, \ldots, S\} \tag{21d}$$

where the subscript $(\cdot)_{k,j|t}$ denotes the j-th scenario predicted k steps ahead of the current time t; ω_j is the probability of occurrence of the j-th disturbance sequence $W_{j|t} = (w_{0,j|t}, \ldots, w_{N-1,j|t})$; $\mathcal{X}$ and $\mathcal{U}$ are the state and input constraints, respectively, that must hold for all uncertainty realizations; L and L_f are the stage and terminal cost functions, respectively; and (21c) enforces the *nonanticipativity constraints*, which ensure that states that branch from the same parent node have the same control input. As shown in [76], these constraints can be written in terms of known matrices $\{\tilde{E}_j\}_{j=1}^{S}$ that impose structure on the vector of control inputs for the jth scenario, i.e., $U_{j|t} = (u_{0,j|t}, \ldots, u_{N-1,j|t})$. To limit the exponential growth in the scenario tree, we use the idea of the robust horizon $N_r < N$ from [75], which stops branching after N_r steps with N_r usually equal to 2 or 3 (often sufficient in practice to achieve constraint satisfaction).

In (21), the control objective is given by a terminal cost of $L_f(x_N) = (\text{CEM}_{\text{sp}} - \text{CEM}_{N|t})^2$, where CEM_{sp} is the desired CEM value. We use a prediction horizon of $N = 5$ and a robust horizon of $N_r = 2$. We consider a set of three scenarios at each time step, mainly $\{(-\hat{w}_{\text{bound}}, -\hat{w}_{\text{bound}}), (0, 0), (\hat{w}_{\text{bound}}, \hat{w}_{\text{bound}})\}$, which correspond to low, middle, and high values for the uncertainty for a tuning parameter $\hat{w}_{\text{bound}}$. The inputs are constrained by the hardware to satisfy $P \in [1.5, 5.0]$ W and $q \in [1.5, 5.0]$ s.l.m. The outputs are constrained by the following limits $T \in [25, 45]$ °C and $I \in [0, 80]$ intensity units. The optimization problem (21) was formulated using CasADi [54] and solved with IPOPT [11] in a receding-horizon fashion.

Although sMPC provides an effective robust control strategy for the APPJ, it poses a significant challenge for CoC design since, to the best of the author's knowledge, there are no known off-the-shelf embedded implementations of (21). As such, sMPC is an ideal test case for our proposed framework since it provides a useful expert control policy that can be imitated by a DNN on embedded hardware.

D. Optimized CoC Design

Now, our goal is to embed the sMPC policy defined by (21) on specialized hardware, mainly an FPGA (i.e., the programmable logic side of the Zybo Z7). We can select CoC design parameters at each stage of the proposed framework—we list several possible parameters in Table II. We selected the following closed-loop performance metrics to define the MOO problem (9) that provides the basis for selecting the complete set of software and hardware design parameters ξ

TABLE II
EXAMPLES OF DESIGN PARAMETERS IN THE COC DESIGN PROCESS

CoC Design Step	Examples
Scenario-based MPC (λ)	prediction horizon (N), robust horizon (N_r), back-off parameters, uncertainty bounds ($\hat{w}_{\text{bound}}$)
DNN Approximation (θ_A)	number of layers (L), number of nodes (H), activation function(s) ($\{\alpha_l\}_{l=1}^{L}$), training/optimizer parameters
Hardware Implementation (FPGA) (γ)	numerical representation (e.g., total/fraction length/bit representation), parallelization options

$$J_1(\xi) = \mathbb{E}\left\{ \sum_{t=0}^{T} \left(\text{CEM}_{\text{sp}} - \text{CEM}_t(\xi, W)\right)^2 \right\} \tag{22a}$$

$$J_2(\xi) = \mathbb{E}\left\{ \sum_{t=0}^{T} \left([T_t(\xi, W) - T_{\max}]^+\right)^2 \right\} \tag{22b}$$

where $W = (w_0, \ldots, w_{T-1})$ is the set of random uncertainty values over the horizon T, $[a]^+ = \max\{a, 0\}$, and the expectation is taken over W. Here, J_1 represents the deviation from the desired setpoint of $\text{CEM}_{\text{sp}} = 1.5$ min and J_2 is a temperature constraint violation metric given $T_{\max} = 45$ °C.

We select the following CoC design parameters and their bounds: maximum possible uncertainty realization $\hat{w}_{1,\text{bound}} \in [0, 10]$, the number of nodes per hidden layer in the DNN $H \in [2, 10]$, and the fixed-point word length when generating code $\text{wl} \in [10, 32]$.

1) Hardware Feasibility Classifier Results: As mentioned in Section III-B, the constraint function $g(\theta_A, \gamma)$ can be learned prior to any optimization routine. As such, we preselect a range of H and wl to learn a binary classifier $\tilde{g}(H, \text{wl}) \rightarrow \{0, 1\}$. Data $\mathcal{D}_g = \{(H, \text{wl}), g(H, \text{wl})\}$ were generated by creating DNNs with randomly initialized weights and passing them through the code generation process $\mathcal{CG}$. Feasibility was recorded with each combination of parameters. Using $\mathcal{D}_g$, a neural-network-based classifier was trained using MATLAB's `fitcnet` function with an 80/20 training/test split.

Remark 3: While the generation of training data for the hardware feasibility classifier is performed independently of the CoC design optimization, the process of generating the data itself can be costly due to the time required to run the code generation step (i.e., the FPGA synthesis). To reduce the cost of gathering training data, optimization methods, such as BO, can be used to determine the boundaries of feasible/infeasible points instead of searching over the entire parameter space of Θ_A and Γ.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

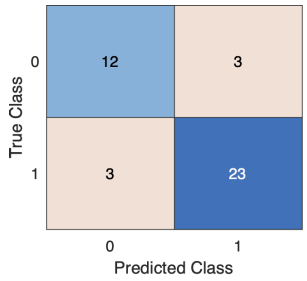

Fig. 6. Confusion matrix of the hardware feasibility classifier. Recall that 0 represents an infeasible CoC design, while 1 indicates a feasible design.

Fig. 6 shows the confusion matrix of the resulting hardware classifier learned from $\mathcal{D}_n$. The confusion matrix shows a reliable estimation of how well we can predict the feasibility of a particular hardware design. As shown in Fig. 6, the classifier accurately predicts with an 85% test accuracy. This hardware classifier can then be used during an optimization routine to estimate which combination of H and wl are feasible. The result is that we effectively reduce the search space prior to performing full CoC design runs.

2) Multiobjective Framework Results: Five replicates of MOBO are used to iteratively find the Pareto frontier according to the two closed-loop performance metrics using a total of 50 iterations. Closed-loop FPGA-in-the-loop simulations of the APPJ are performed in MATLAB. DNNs are trained with `feedforwardnet` and $n_s = 2000$. Again, the choice of n_s was done empirically. However, a nice feature of our CoC framework is that the closed-loop validation process is able to "catch" if poor performance is consistently achieved for a variety of design parameters ξ, which, in turn, may be indicative of a poor training dataset. Since the CoC optimization is performed offline (up until the real-system query), one can always increase the size of n_s at the cost of longer data generation and training time. Results from the closed-loop simulations are passed into the MOBO framework implemented in Python using Ax [36]. MOBO is compared to random search (RS) using a quasi-random Sobol sampling strategy [77], which is a common benchmark for DFO methods. Each strategy was initialized with one known, successful design, and one Sobol-selected design.

Fig. 7 illustrates the observed closed-loop performance metrics over the five replicates of MOBO and RS. In Fig. 7, the blue circles indicate data points gathered from MOBO, while red squares indicate data points gathered from RS. Black dashed lines indicate "objective thresholds," which are used in Ax to "constrain" the search space to produce values within a desired region-of-interest. Objective thresholds are chosen such that the closed-loop performance metrics are representative of valuable or practical control policies. In other words, in the case of the thermal dose metric, if the metric exceeds 100 ($\approx \text{CEM}_{\text{sp}} \times 70$ time steps), then this would mean that the suggested CoC policy will take longer than the maximum treatment time allowed and/or is incapable of achieving the desired thermal dose. In the case of the temperature constraint metric, if the metric exceeds 80 ($\approx 1\ ^\circ\text{C} \times 80$ time steps or $10\ ^\circ\text{C} \times 8$ time steps), then the policy is considered too dangerous as it violates the constraint too often or at too high magnitude. Objective thresholds can be chosen since MOBO produces a surrogate model that relates the design parameters to the performance metrics. The posterior model can be used to estimate parameters that will produce performance metrics that are likely to be in the defined region-of-interest. Since the surrogate model in MOBO is probabilistic in nature, CoC design parameters suggested by MOBO may still fall outside of the region-of-interest in the initial iterations of MOBO. Note that the RS has no notion of the objective threshold since no surrogate model is created based on the previously observed data. As such, RS explores significantly more designs outside of the region-of-interest due to a less constricted parameter space. The left subfigure shows all data encountered, while the right subfigure shows a zoomed-in version with a truncated x-axis, and it illustrates how RS encounters significantly more points with a thermal dose metric greater than 100.

Furthermore, Fig. 8 shows the HV evolution over the 50 iterations of each method (MOBO in blue and RS in red). The solid lines indicate the mean over five replicates, whereas the shaded region indicates one standard error. Recall that the HV is a measure of the quality of the Pareto frontier. As seen in Fig. 8, the two methods begin in a similar fashion at suboptimal Pareto frontier estimations over the first two iterations, but MOBO quickly diverges within the immediate next few iterations. MOBO's increase in HV value in few iterations is due to its ability to intelligently explore the design space in search of the optimal tradeoff between the two performance metrics. Meanwhile, RS is a naïve approach that explores many options that are not expected to improve the HV, i.e., designs are selected with no knowledge of the outcome. As such, it takes RS many more iterations to reach an HV close to that of a "converged" MOBO.

Finally, Fig. 9 shows the closed-loop trajectories of the states related to J_1 and J_2 (CEM and T, respectively). We use Fig. 9 to illustrate the performances of various CoC designs that are encountered at several snapshots of MOBO. Using Fig. 9, we can see the variety of control policies that are encountered, and how the observed Pareto frontier evolves over time. From top to bottom, we show snapshots of one replicate of MOBO at Iterations 5, 15, and 25. From left to right, the subfigures in the left column are the closed-loop trajectories of CEM for selected designs, the subfigures in the middle column are the closed-loop trajectories of surface temperature for selected designs, and the subfigures in the right column are the observed metrics from the designs encountered up until that iteration. We selected designs that a practitioner may select based on the application's needs. A "utopia" design is determined based on the lowest combination (scaled sum) of performance metric values

$$\Psi_u = \hat{J}_1 + \hat{J}_2$$

where $\hat{J}_i$ are scaled values of J_i. Bounds of the scaling are fixed to $[20, 100]$ for J_1 and $[0, 60]$ for J_2. A "control performance preferred" design is determined based on a weighted combination of J_1 and J_2

$$\Psi_p = \tau \hat{J}_1 + \frac{1}{\tau} \hat{J}_2$$

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

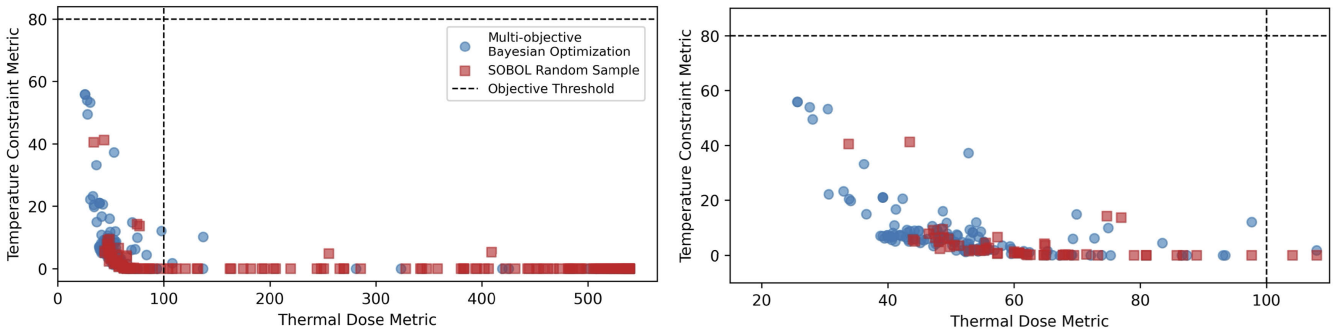

Fig. 7. Observed closed-loop performance metrics of plasma treatments during five replicates each of MOBO and RS via SOBOL sampling. Blue circles indicate the metrics observed during MOBO; red squares indicate the metrics observed during RS. Black dashed lines indicate "constraints" on the closed-loop performance metrics that are used to guide parameter suggestions to the region-of-interest. The left figure shows all data encountered in all optimization routines, while the right figure illustrates a zoomed-in version (truncating the upper x-axis value at 120). Note that RS has no notion of the objective threshold since no surrogate model is created based on previously observed data. As such, RS explores significantly more designs outside of the region-of-interest.

TABLE III

CONFIGURATIONS AND CLOSED-LOOP METRICS FOR REAL-TIME CONTROL EXPERIMENTS WITH THE APPJ TESTBED. METRICS ARE REPORTED AS THE MEAN ± THE STANDARD ERROR OF FIVE REPLICATES

		Control Policy Parameters				Closed-loop Performance Metrics	
Configuration	Description	Hidden Nodes	Hidden Layers	Fixed Point Word Length	Loop Unrolling	Thermal Dose	Temperature Constraint
(i)	Control Performance Preferred	5	2	13	UnrollLoops	35.66 ± 0.59	2.12 ± 0.43
(ii)	Constraint Satisfaction Preferred	8	2	18	LoopNone	39.53 ± 4.19	1.07 ± 0.40
(iii)	Utopia	5	3	20	LoopNone	33.67 ± 2.03	1.78 ± 0.39

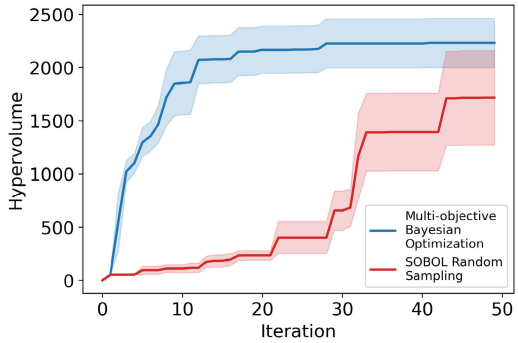

Fig. 8. Hypervolume improvement of five replicates each of MOBO (blue) and RS via SOBOL sampling (red) for the CoC design for APPJs. Solid lines indicate the mean hypervolume and the shaded regions indicate one standard error. MOBO on average reaches a higher hypervolume overall and earlier than random sampling, which indicates that a meaningful Pareto frontier is realized in fewer iterations of MOBO than random sampling.

where $\tau = 3$. A "constraint satisfaction preferred" design is determined similarly, but with the weights switched, i.e.,

$$\Psi_s = \frac{1}{\tau}\hat{J}_1 + \tau \hat{J}_2.$$

Designs are selected in this manner to avoid overly extreme controller designs. Utopia designs are denoted by the color green (and solid lines in the trajectory plots), control performance preferred designs are denoted by the color orange (dotted-dashed lines), and constraint satisfaction preferred designs are denoted by the color brown. In addition, if any selected designs were the first artificial point, then they are colored purple. At Iteration 5, MOBO has selected designs that improve the Pareto frontier such that a utopia design and control performance preferred design are different from the first design. The utopia point provides an indication of design with the most even tradeoff between the different metrics and can also partially indicate the quality of the Pareto frontier. At Iteration 15, more designs have been explored such that new designs for the more extreme designs can be observed. At Iteration 25, several more designs were explored but had little value in finding the Pareto optimal points and, as a result, had no significant changes in Iteration 15.

E. Real-Time Experiment Results

Finally, to show utility in the real system, we selected three control policy designs from the Pareto frontier generated in simulation. Note that, for ease of testing, we chose to modify H, L, wl, and loop where a loop is a categorical option in the code generation process that determines whether or not to parallelize certain matrix computations. We used MOBO offline to search for optimal CoC designs and transfer the designs to real-time experiments. In principle, these optimal design parameters obtained offline are sufficient for the experiments as they primarily describe the capability of the CoC policy in: 1) accurately representing the sMPC law and 2) being feasibly implemented on the hardware device. Fig. 10 shows the observed data from three replicates of MOBO for the new set of design parameters. Fig. 10 shows a similar

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

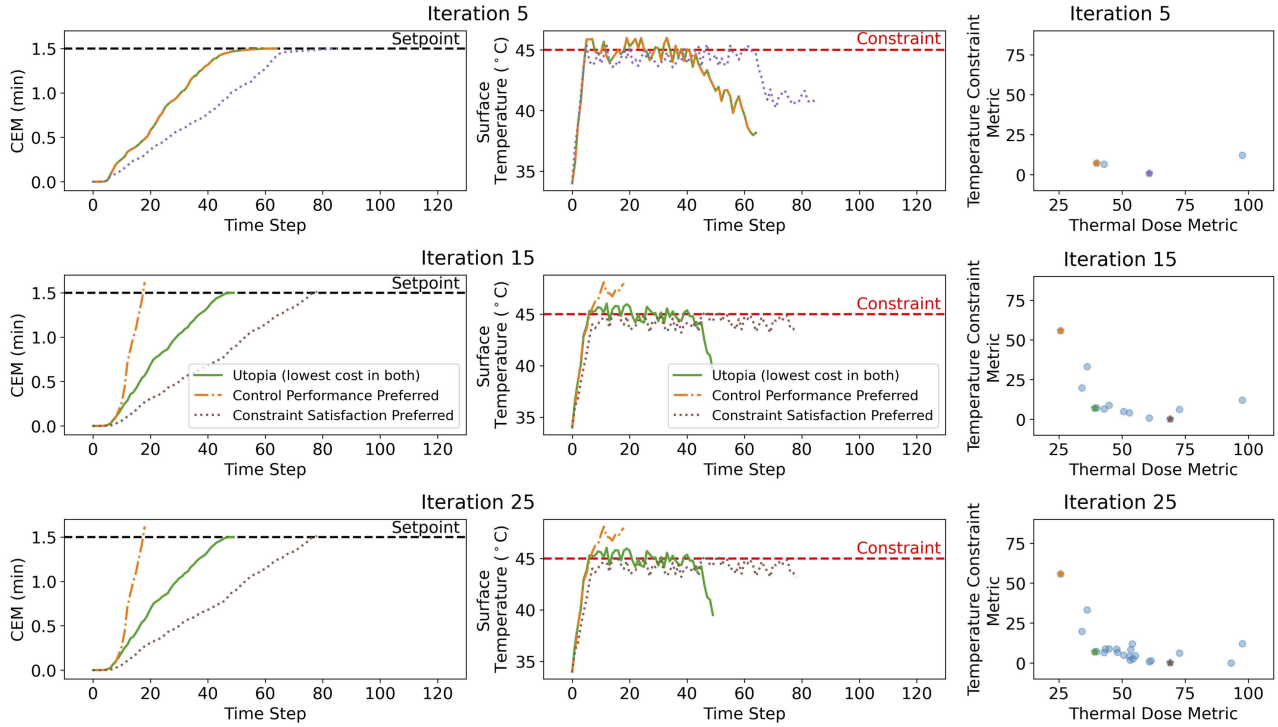

Fig. 9. Closed-loop trajectories of CEM (left column) and surface temperature (middle column) at snapshots (at Iterations 5, 15, and 25) of one replicate of MOBO. Selected designs (colored stars) were determined from the observed data (blue circles) in the metric space (right column). The selected designs correspond to designs that an engineer may select based on the needs of a particular application. The "utopia" point/design (green solid) is selected based on the lowest combination (scaled summation) of the metric values. The "control performance preferred" point (orange dotted-dashed) is selected based on a weighted combination of the metric values where the thermal dose metric is weighted three times more than the temperature constraint metric. The "constraint satisfaction preferred" point (brown dotted) is selected based on a weighted combination of the metric values where the temperature constraint metric is weighted three times more than the thermal dose metric. In the CEM figures, the black dashed line represents the desired thermal dose. In the surface temperature figures, the red dashed line represents the constraint.

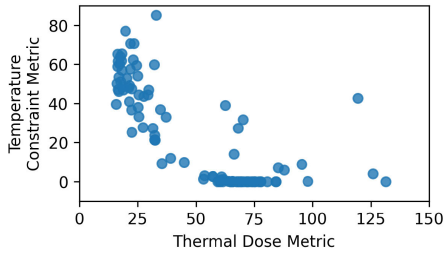

Fig. 10. Observed performance metrics during three replicates of MOBO for CoC design based on design parameters chosen for the experimental case study. Modifying different design parameters still shows a similar tradeoff between the two closed-loop performance metrics.

tradeoff as in Fig. 7 even with different design parameters. Real-time experiments using the CoC design parameters to create embedded control policies for the APPJ were performed in triplicates. Table III describes the controller parameters used and the closed-loop metrics obtained on the APPJ testbed. Due to the significant plant-model mismatch between the linear models and the true system, the original sMPC was tuned based on a newly identified LTI model on the day of experiments to achieve appropriate control performance before testing CoC designs.[5] From Table III, we can see several tradeoffs between control performance, hardware utilization, and constraint satisfaction. A performance-dominated CoC design [i.e., Configuration 1)] is typically comprised of a small-scale DNN that can achieve the desired CEM dose quickly at the expense of more constraint violations. A constraint-dominated CoC design [i.e., Configuration 2)] is typically comprised of a larger width DNN with a larger fixed-point word length. This often leads to a more representative DNN that has fewer constraint violations. Finally, a mixed CoC design [i.e., Configuration 3)] can offer a balance of the two extremes.

[5]New system parameters are provided in Appendix A.

VII. Conclusion and Future Work

This article presented an end-to-end CoC design framework for the implementation of arbitrary control policies on arbitrary hardware. We argued for deep learning as a unifying template that connects the hardware and software aspects of embedded control design. Furthermore, we presented a BO framework for CoC design that can account for the multiobjective nature of the control design problem, categorical design spaces, and minimal interactions with the expensive design process. We demonstrated the proposed CoC design framework for CAP processing of biomaterials in closed-loop simulation studies and real-time experiments. The framework was able to efficiently and systematically determine tradeoffs in the CoC design process, resulting in adequate estimation of the Pareto frontier in only a few design iterations. Future work will focus on extending the framework to online adaptation of control policies and guaranteeing the robustness (i.e., safety) of the resulting controllers.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

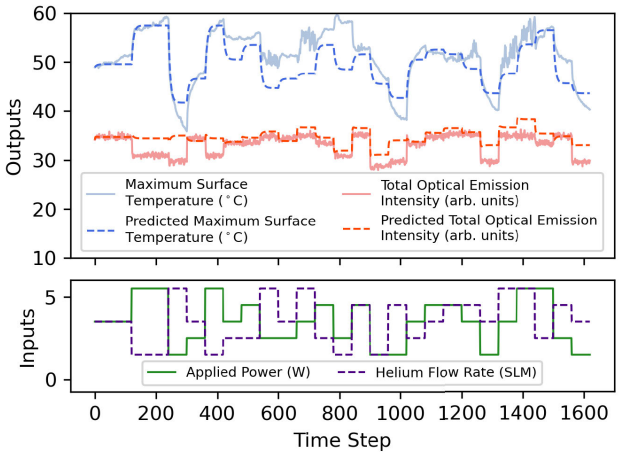

Fig. 11. Comparison of measured APPJ outputs and the model used in sMPC [see (25)]. A new model was learned from new experimental data since the configuration of the APPJ had changed since the data collection for (23) and (24).

APPENDIX A
SUBSPACE IDENTIFICATION FOR THE APPJ TESTBED

As indicated in the main text, we used subspace identification [71] to identify discrete-time state-space models from open-loop data of the APPJ testbed. For the closed-loop simulations described in Section VI-D, we used two sets of identified matrices to simulate plant-model mismatch. The following matrices were used in defining the control model $\hat{f}(x,u)$:

$$A = \begin{bmatrix} 0.903 & 0.018 \\ 0.132 & 0.243 \end{bmatrix}, \quad B = \begin{bmatrix} 0.581 & -0.220 \\ 2.674 & -1.131 \end{bmatrix}. \tag{23}$$

The following matrices were used in defining the plant model (19):

$$A = \begin{bmatrix} 0.888 & 0.055 \\ 0.094 & 0.283 \end{bmatrix}, \quad B = \begin{bmatrix} 0.503 & -0.174 \\ 2.764 & -1.037 \end{bmatrix}. \tag{24}$$

In Section VI-E, a new state-space model was identified for sMPC. The matrices for $\hat{f}(x,u)$ are given as follows:

$$A = \begin{bmatrix} 0.845 & 0.016 \\ -0.060 & 0.358 \end{bmatrix}, \quad B = \begin{bmatrix} 0.450 & -0.164 \\ 0.713 & 0.567 \end{bmatrix}. \tag{25}$$

Furthermore, the nominal operating conditions were given as $[P^s, q^s]^\top = [3.5\text{ W},\ 3.5\text{ s.l.m.}]^\top$ and $[T^s, I^s]^\top = [49.6\text{ °C},\ 29.7\text{ arb. units}]^\top$. Fig. 11(b) shows a comparison between the open-loop data collected (via multistep tests) and the model learned from the open-loop data.

APPENDIX B
COMPILATION SETTINGS FOR GENERATING EMBEDDED CONTROL POLICIES

This section describes the settings used in generating code for hardware-based control policies. Most tools were used with their default settings with the exception of those detailed in this section.

A. CasADi Code Generation

As mentioned in the main text, CasADi provides a `generate()` function to create a C-code representation of the optimization problem defined with CasADi's symbolic syntax. The optimization problem (17) from the illustrative example in Section V is formulated using CasADi's symbolic syntax and used to generate a functional form of the control law

$$u_t^\star = \pi_{\text{mpc}}(x_t) \tag{26}$$

where a call to π_{mpc} solves the implicit optimization problem defined by (17). The variable in which π_{mpc} is stored is used with the `generate(filename, opts)` function with these additional options (defined in as the structure `opts`):

1) `'main': true`;
2) `'mex': true`;
3) `'with_header': true`.

Then, the C-code generated under the filename variable `filename` is converted to the `MEX` function for use as the `MATLAB` function using the following command (within `MATLAB` Command Window): `mex filename -largeArrayDims -outdir codegen/mex/mpc/ -output mpc_mex`.

B. MPT-3 Code Generation

As mentioned in the main text, MPT-3 provides several methods to generate C-code for control policies defined using MPT-3 syntax. In this work, we use the method that involves the use of the `MATLAB Coder` toolbox. The procedure to do so is as follows.

1) Define a control policy using the `MPCController` object, which takes `system` and `horizon` as arguments, where `system` is the linear model to be used in the dynamics constraints and `horizon` is the prediction horizon of the optimal control problem: `c = MPCController(system, horizon)`.
2) Convert the control policy into an explicit control policy using the `toExplicit()` function: `ec = c.toExplicit()`.
3) Convert the explicit control policy to a functional representation of the control law: `ec.toMatlab(mfile, 'primal', 'obj')`, where the `mfile` is an m-code representation of the control law

$$u_t^\star = \pi_{\text{empc}}(x_t) \tag{27}$$

where the evaluation of π_{empc} searches for the appropriate gain from the PWA function that defines the explicit control policy.
4) Use `MATLAB Coder` to create a `MEX` function: `coder mfile -args {zeros(nx, 1)}`, where `nx` is the number of states.

C. DNN Code Generation

The DNN can be converted to C-code through a similar method as the final step of the MPT-3 method. The DNN

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

was created using the `feedforwardnet` function, which is a part of the Deep Learning toolbox. In addition, the Deep Learning toolbox provides a function `genFunction` that converts a neural network to the `MATLAB` function, i.e., an m-code representation of the DNN control policy

$$\hat{u}_t^\star = \pi_{\text{dnn}}(x_t) \tag{28}$$

where the evaluation of π_{dnn} is the forward pass of the learned DNN. The DNN is converted with `genFunction`: `genFunction(net, mfile, 'MatrixOnly', 'yes')`, where `net` is the DNN object generated after calling `feedforwardnet` and `mfile` is the filename of the m-code function that is generated. Finally, the same call to `MATLAB Coder` from step 4 of the MPT-3 code generation is called to create the C-code and `MEX` function.

REFERENCES

[1] S. Kato et al., "Autoware on board: Enabling autonomous vehicles with embedded systems," in *Proc. ACM/IEEE 9th Int. Conf. Cyber-Phys. Syst. (ICCPS)*, Apr. 2018, pp. 287–296.

[2] D. Mazzei, G. Montelisciani, G. Baldi, and G. Fantoni, "Changing the programming paradigm for the embedded in the IoT domain," in *Proc. IEEE 2nd World Forum Internet Things (WF-IoT)*, Dec. 2015, pp. 239–244.

[3] C. Hao et al., "FPGA/DNN co-design: An efficient design methodology for IoT intelligence on the edge," in *Proc. 56th Annu. Des. Autom. Conf.*, 2019, pp. 1–6.

[4] A. Moradkhani, A. Broumandnia, and S. J. Mirabedini, "A portable medical device for detecting diseases using probabilistic neural network," *Biomed. Signal Process. Control*, vol. 71, Jan. 2022, Art. no. 103142.

[5] M. Farahi et al., "Beat-to-beat fetal heart rate analysis using portable medical device and wavelet transformation technique," *Heliyon*, vol. 8, no. 12, Dec. 2022, Art. no. e12655.

[6] G. F. Franklin, J. D. Powell, and M. L. Workman, *Digital Control of Dynamic Systems*, vol. 3. Menlo Park, CA, USA: Addison-Wesley, 1998.

[7] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—An open-source framework for automatic control and dynamic optimization," *Optim. Control Appl. Methods*, vol. 32, no. 3, pp. 298–312, May 2011.

[8] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)," *Optim. Eng.*, vol. 20, no. 3, pp. 769–809, Sep. 2019.

[9] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari, "FORCES NLP: An efficient implementation of interior-point methods for multistage nonlinear nonconvex programs," *Int. J. Control*, vol. 93, no. 1, pp. 13–29, Jan. 2020.

[10] R. Quirynen, M. Vukov, M. Zanon, and M. Diehl, "Autogenerating microsecond solvers for nonlinear MPC: A tutorial using ACADO integrators," *Optim. Control Appl. Methods*, vol. 36, no. 5, pp. 685–704, Sep. 2015.

[11] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Math. Program.*, vol. 106, no. 1, pp. 25–57, 2006.

[12] S. Richter, C. N. Jones, and M. Morari, "Computational complexity certification for real-time MPC with input constraints based on the fast gradient method," *IEEE Trans. Autom. Control*, vol. 57, no. 6, pp. 1391–1403, Jun. 2012.

[13] W. H. Wolf, "Hardware-software co-design of embedded systems," *Proc. IEEE*, vol. 82, no. 7, pp. 967–989, Jul. 1994.

[14] G. De Michell and R. K. Gupta, "Hardware/software co-design," *Proc. IEEE*, vol. 85, no. 3, pp. 349–365, Feb. 1997.

[15] Y. Nishikawa, N. Sannomiya, T. Ohta, and H. Tanaka, "A method for auto-tuning of PID control parameters," *Automatica*, vol. 20, no. 3, pp. 321–332, May 1984.

[16] C. C. Hang, K. J. Astrom, and Q. G. Wang, "Relay feedback auto-tuning of process controllers—A tutorial review," *J. Process Control*, vol. 12, no. 1, pp. 143–162, Jan. 2002.

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

[18] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, "Safe model-based reinforcement learning with stability guarantees," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.

[19] M. Zanon and S. Gros, "Safe reinforcement learning using robust MPC," *IEEE Trans. Autom. Control*, vol. 66, no. 8, pp. 3638–3652, Sep. 2020.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[21] M. Neumann-Brosig, A. Marco, D. Schwarzmann, and S. Trimpe, "Data-efficient autotuning with Bayesian optimization: An industrial control study," *IEEE Trans. Control Syst. Technol.*, vol. 28, no. 3, pp. 730–740, May 2020.

[22] C. König, M. Turchetta, J. Lygeros, A. Rupenyan, and A. Krause, "Safe and efficient model-free adaptive control via Bayesian optimization," in *Proc. IEEE Int. Conf. Rob. Autom.*, May 2021, pp. 9782–9788.

[23] J. A. Paulson, F. Sorourifar, and A. Mesbah, "A tutorial on derivative-free policy learning methods for interpretable controller representations," in *Proc. Amer. Control Conf. (ACC)*, May 2023, pp. 1295–1306.

[24] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of Bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[25] M. Zhu, A. Bemporad, and D. Piga, "Preference-based MPC calibration," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2021, pp. 638–645.

[26] M. Forgione, D. Piga, and A. Bemporad, "Efficient calibration of embedded MPC," *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 5189–5194, 2020.

[27] M. Zhu, D. Piga, and A. Bemporad, "C-GLISp: Preference-based global optimization under unknown constraints with applications to controller calibration," *IEEE Trans. Control Syst. Technol.*, vol. 30, no. 5, pp. 2176–2187, Sep. 2022.

[28] T. Parisini and R. Zoppoli, "A receding-horizon regulator for nonlinear systems and a neural approximation," *Automatica*, vol. 31, no. 10, pp. 1443–1451, 1995.

[29] S. Chen et al., "Approximating explicit model predictive control using constrained neural networks," in *Proc. Amer. Control Conf.*, 2018, pp. 1520–1527.

[30] K. J. Chan, G. Makrygiorgos, and A. Mesbah, "Towards personalized plasma medicine via data-efficient adaptation of fast deep learning-based MPC policies," in *Proc. Amer. Control Conf.*, 2023, pp. 2769–2775.

[31] J. Drgoňa, K. Kiš, A. Tuor, D. Vrabie, and M. Klaučo, "Differentiable predictive control: Deep learning alternative to explicit model predictive control for unknown nonlinear systems," *J. Process Control*, vol. 116, pp. 80–92, Aug. 2022.

[32] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective Bayesian optimization," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1492–1501.

[33] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Struct. Multidiscip. Optim.*, vol. 26, no. 6, pp. 369–395, 2004.

[34] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms—A comparative case study," in *Proc. Int. Conf. Parallel Probl. Solving Nat.*, 1998, pp. 292–301.

[35] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*, vol. 1. Berlin, Germany: Springer, 2006.

[36] E. Bakshy et al., "AE: A domain-agnostic platform for adaptive experimentation," in *Proc. Conf. Neural Inf. Process. Syst.*, 2018, pp. 1–8.

[37] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 7553.

[38] A. D. Bonzanini, J. A. Paulson, G. Makrygiorgos, and A. Mesbah, "Fast approximate learning-based multistage nonlinear model predictive control using Gaussian processes and deep neural networks," *Comput. Chem. Eng.*, vol. 145, Feb. 2021, Art. no. 107174.

[39] S.-I. Amari, "Backpropagation and stochastic gradient descent method," *Neurocomputing*, vol. 5, nos. 4–5, pp. 185–196, 1993.

[40] D. Justus, J. Brennan, S. Bonner, and A. S. McGough, "Predicting the computational cost of deep learning models," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2018, pp. 3873–3882.

[41] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011.

[42] A. Suardi, E. C. Kerrigan, and G. A. Constantinides, "Fast FPGA prototyping toolbox for embedded optimization," in *Proc. Eur. Control Conf. (ECC)*, Jul. 2015, pp. 2589–2594.

[43] S. Lucia, D. Navarro, Ó. Lucía, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Trans. Ind. Informat.*, vol. 14, no. 1, pp. 137–145, Jan. 2018.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[44] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[45] A. J. Kleywegt, A. Shapiro, and T. Homem-de Mello, "The sample average approximation method for stochastic discrete optimization," *SIAM J. Optim.*, vol. 12, no. 2, pp. 479–502, 2002.

[46] A. Smola and P. Bartlett, "Sparse greedy Gaussian process regression," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 13, 2000, pp. 1–7.

[47] J. Qui nonero-Candela and C. E. Rasmussen, "A unifying view of sparse approximate Gaussian process regression," *J. Mach. Learn. Res.*, vol. 6, pp. 1939–1959, Dec. 2005.

[48] B. Lei et al., "Bayesian optimization with adaptive surrogate models for automated experimental design," *Npj Comput. Mater.*, vol. 7, no. 1, p. 194, Dec. 2021.

[49] R. Moriconi, M. P. Deisenroth, and K. S. S. Kumar, "High-dimensional Bayesian optimization using low-dimensional feature spaces," *Mach. Learn.*, vol. 109, nos. 9–10, pp. 1925–1943, Sep. 2020.

[50] S. Daulton, M. Balandat, and E. Bakshy, "Parallel Bayesian optimization of multiple noisy objectives with expected hypervolume improvement," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021, pp. 2187–2200.

[51] G. Makrygiorgos, A. D. Bonzanini, V. Miller, and A. Mesbah, "Performance-oriented model learning for control via multi-objective Bayesian optimization," *Comput. Chem. Eng.*, vol. 162, Jun. 2022, Art. no. 107770.

[52] D. da Silva, "Proprietades geraes," *J. de L'Ecole Polytechnique, Cah*, vol. 30, no. 1, pp. 1–11, 1854.

[53] Y. Wang and S. Boyd, "Fast model predictive control using online optimization," *IEEE Trans. Control Syst. Technol.*, vol. 18, no. 2, pp. 267–278, Mar. 2010.

[54] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi: A software framework for nonlinear optimization and optimal control," *Math. Program. Comput.*, vol. 11, no. 1, pp. 1–36, Mar. 2019.

[55] A. Alessio and A. Bemporad, *A Survey on Explicit Model Predictive Control*. Berlin, Germany: Springer, 2009, pp. 345–369.

[56] M. Herceg, M. Kvasnica, C. N. Jones, and M. Morari, "Multi-parametric toolbox 3.0," in *Proc. Eur. Control Conf. (ECC)*, 2013, pp. 502–510. [Online]. Available: http://control.ee.ethz.ch/~mpt

[57] P. K. Chu, J. Y. Chen, L. P. Wang, and N. Huang, "Plasma-surface modification of biomaterials," *Mater. Sci. Eng., R, Rep.*, vol. 36, nos. 5–6, pp. 143–206, Mar. 2002.

[58] M. G. Kong et al., "Plasma medicine: An introductory review," *New J. Phys.*, vol. 11, no. 11, Nov. 2009, Art. no. 115012.

[59] K.-D. Weltmann and T. von Woedtke, "Plasma medicine—Current state of research and medical application," *Plasma Phys. Controlled Fusion*, vol. 59, no. 1, Jan. 2017, Art. no. 014031.

[60] M. Laroussi, "Plasma medicine: A brief introduction," *Plasma*, vol. 1, no. 1, pp. 47–60, Feb. 2018.

[61] D. G. Petlin, S. I. Tverdokhlebov, and Y. G. Anissimov, "Plasma treatment as an efficient tool for controlled drug release from polymeric materials: A review," *J. Controlled Release*, vol. 266, pp. 57–74, Nov. 2017.

[62] Y. Gorbanev et al., "Combining experimental and modelling approaches to study the sources of reactive species induced in water by the COST RF plasma jet," *Phys. Chem. Chem. Phys.*, vol. 20, no. 4, pp. 2797–2808, 2018.

[63] X. Lu, M. Laroussi, and V. Puech, "On atmospheric-pressure non-equilibrium plasma jets and plasma bullets," *Plasma Sources Sci. Technol.*, vol. 21, no. 3, Jun. 2012, Art. no. 034005.

[64] L. Lin, D. Yan, T. Lee, and M. Keidar, "Self-adaptive plasma chemistry and intelligent plasma medicine," *Adv. Intell. Syst.*, vol. 4, no. 3, Mar. 2022, Art. no. 2100112.

[65] D. Gidon, B. Curtis, J. A. Paulson, D. B. Graves, and A. Mesbah, "Model-based feedback control of a kHz-excited atmospheric pressure plasma jet," *IEEE Trans. Radiat. Plasma Med. Sci.*, vol. 2, no. 2, pp. 129–137, Mar. 2018.

[66] A. Yang, X. Wang, M. Rong, D. Liu, F. Iza, and M. G. Kong, "1-D fluid model of atmospheric-pressure RF $He+O_2$ cold plasmas: Parametric study and critical evaluation," *Phys. Plasmas*, vol. 18, no. 11, Nov. 2011, Art. no. 113503.

[67] C. Chen et al., "A model of plasma-biofilm and plasma-tissue interactions at ambient pressure," *Plasma Chem. Plasma Process.*, vol. 34, no. 3, pp. 403–441, Apr. 2014.

[68] L. Lin and M. Keidar, "A map of control for cold atmospheric plasma jets: From physical mechanisms to optimizations," *Appl. Phys. Rev.*, vol. 8, no. 1, Mar. 2021, Art. no. 011306.

[69] A. Mesbah and D. B. Graves, "Machine learning for modeling, diagnostics, and control of non-equilibrium plasmas," *J. Phys. D, Appl. Phys.*, vol. 52, no. 30, Jul. 2019, Art. no. 30LT02.

[70] L. Lin and M. Keidar, "Machine learning controlled self-adaptive plasma medicine," in *Proc. IEEE Int. Conf. Plasma Sci. (ICOPS)*, Dec. 2020, p. 561.

[71] P. Van Overschee and B. De Moor, *Subspace Identification for Linear Systems: Theory—Implementation—Applications*. Berlin, Germany: Springer, 2012.

[72] D. Gidon, D. B. Graves, and A. Mesbah, "Effective dose delivery in atmospheric pressure plasma jets for plasma medicine: A model predictive control approach," *Plasma Sources Sci. Technol.*, vol. 26, no. 8, Jul. 2017, Art. no. 085005.

[73] S. A. Sapareto and W. C. Dewey, "Thermal dose determination in cancer therapy," *Int. J. Radiat. Oncol. Biol. Phys.*, vol. 10, no. 6, pp. 787–800, Apr. 1984.

[74] D. Bernardini and A. Bemporad, "Scenario-based model predictive control of stochastic constrained linear systems," in *Proc. 48th IEEE Conf. Decis. Control (CDC)*, Dec. 2009, pp. 6333–6338.

[75] S. Lucia, T. Finkler, and S. Engell, "Multi-stage nonlinear model predictive control applied to a semi-batch polymerization reactor under uncertainty," *J. Process Control*, vol. 23, no. 9, pp. 1306–1319, 2013.

[76] E. Klintberg, J. Dahl, J. Fredriksson, and S. Gros, "An improved dual Newton strategy for scenario-tree MPC," in *Proc. IEEE 55th Conf. Decis. Control*, 2016, pp. 3675–3681.

[77] I. M. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Comput. Math. Math. Phys.*, vol. 7, no. 4, pp. 86–112, Jan. 1967.

Authorized licensed use limited to: Univ of Calif Berkeley. Downloaded on September 29,2024 at 22:23:09 UTC from IEEE Xplore. Restrictions apply.