INL/RPT-23-75114
Revision 000

Protecting and Defending against Autonomous Control Systems and Digital Twin Cyber Attacks

Response Strategy for Hyperparameter attacks of Digital Twin Machine Learning Models in Nuclear Power Plants

September 2023

M3CT-23IN1105035

Idaho National Laboratory
Chris Spirito

Georgia Institute of Technology
Dr. Fan Zhang
Dr. Md. Musabbir Hossain

Idaho National Laboratory

INL is a U.S. Department of Energy National Laboratory operated by Batelle Energy Alliance, LLC

INL/
Revision 000

Protecting and Defending against Autonomous Control Systems and Digital Twin Cyber Attacks

M3CT-23IN1105035

Idaho National Laboratory
Chris Spirito

Georgia Institute of Technology
Dr. Fan Zhang
Dr. Md. Musabbir Hossain

September 2023

Idaho National Laboratory
Idaho Falls, Idaho 83415

http://www.inl.gov

Prepared for the
U.S. Department of Energy
Office of Nuclear Engineering
Under DOE Idaho Operations Office
Contract DE-AC07-05ID14517

Page intentionally left blank

Page intentionally left blank.

CONTENTS

FIGURES

Page intentionally left blank.

ACRONYMS

API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CVXPY	Convex Optimization
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
LQR	Linear Quadratic Regulator
ML	Machine Learning
NLP	Natural Language Processing
NPP	Nuclear Power Plant
NMT	Neural Machine Translation
PLC	Programmable Logic Controller
Q	State Cost
R	Control Effort Cost
SciPy	Scientific Python
TPU	Tensor Processing Unit

Page intentionally left blank.

1. Response Strategy for Hyperparameter attacks of Digital Twin ML Models in NPPs

Navigating through the complex tapestry of technological advancements, "Response Strategy for Hyperparameter attacks of Digital Twin Machine Learning Model in Nuclear Power Plants" stands at the intersection of cybersecurity and nuclear power plant operations, embarking on a journey through the intricacies of securing digital twins against malicious cyber activities. As nuclear power plants progressively integrate digital twin technology and machine learning models to optimize operations and ensure system reliability, they inadvertently expose themselves to a new spectrum of vulnerabilities, notably in the realm of hyperparameter attacks. Hyperparameters, integral in machine learning model tuning and optimal performance of digital twins, have emerged as a target for adversaries aiming to destabilize the predictive capabilities and therefore, the operational accuracy of these digital entities within critical infrastructures like nuclear plants. This paper, therefore, meticulously threads the needle through the development of a robust response strategy, poised to shield these digital reflections against calculated hyperparameter manipulations, ensuring that the digital twin can effectively and securely function as a reliable proxy for its physical counterpart. The ensuing sections delve into the orchestrated maelstrom of multi-rate time-changing intelligent coordinated hyperparameter attacks and the implementation of event-triggered predictive control, laying down a structured, predictive, and responsive framework that safeguards the nexus where the digital and physical realms of nuclear power plants coalesce.

The operational integrity of digital twins in nuclear power plants depends critically on the security of machine learning hyperparameters. This study makes two different contributions. First, a decision-based idea known as a multi-rate time changing intelligent coordinated hyperparameter attack is put forth. In this attack, many hyperparameters are repeatedly changed using both random and intelligent optimal techniques by the attacker. These assaults introduce varied rates at different attack steps, compromise various amounts of hyperparameters, and improve stealth and flexibility. Second, a technique is developed for event triggered predictive control to rapidly respond to potential hyperparameter attacks. This control integrates a sliding window framework, retaining a history of previous data points and employing linear regression to predict the next data point from the current dataset. The control gain K is determined using the Lyapunov-Krasovskii method, and subsequently, an action is developed. Finally, the outcome of the simulation demonstrates the viability of the proposed method for defending nuclear power plant digital twins from hyperparameter attacks.

1.1. Introduction

A hyperparameter attack on digital twin machine learning models refers to malicious attempts to manipulate the hyperparameters of the model in a way that affects its performance, prediction, or decision-making process. The digital twin concept, in the context of machine learning, denotes a digital replica of a physical asset, system, or process that is used to understand, predict, and optimize performance. By compromising the hyperparameters, adversaries can manipulate the digital twin model's learning process, potentially leading to severe consequences, especially when these models are employed in critical real-world systems.

While cyber-attacks on critical infrastructure have been a topic of concern for many years, the advent of sophisticated machine learning applications in such systems, like those in nuclear power plants, presents a new set of challenges. Over the last decade, with the increase in digitization and adoption of machine learning in nuclear reactors' management and safety systems, there's been a growing concern about hyperparameter attacks. Adversaries, realizing the potential fallout of manipulating these models, view it as an avenue to cause significant disruption, potentially leading to incorrect safety protocols, system malfunctions, or even catastrophic events. Instances in the past, albeit limited, have shown that when these hyperparameters are maliciously tweaked, the digital twin models can produce erroneous outputs that could compromise the safety of the entire facility.

To counter the threats posed by hyperparameter attacks, researchers and engineers are leaning towards robust and adaptive control strategies. One such approach is adaptive predictive control, which constantly adjusts and updates the system's behavior based on real-time inputs and forecasts. By doing so, it can potentially identify anomalies in the model's predictions or outputs that may result from hyperparameter tampering. Furthermore, the integration of event-triggered control can be pivotal. Unlike traditional control mechanisms that operate on a set schedule, event-triggered control operates when specific conditions or 'events' are met. This means the system remains dormant until an anomaly, like a hyperparameter attack, is detected, upon which corrective actions are immediately taken. Together, these strategies offer a resilient and responsive defense mechanism against hyperparameter attacks, ensuring the integrity and safety of systems like nuclear power plants.

1.2. Modeling of hyperparameter attack

A digital twin is established in local server, which is connected with GPWR laptop. The GPWR is a WSC full-scope high-fidelity Generic Nuclear Power Plant Simulators for a Pressurized Water Reactor (GPWR). In this research, the following five hyperparameters of machine learning model are attacked as follows:

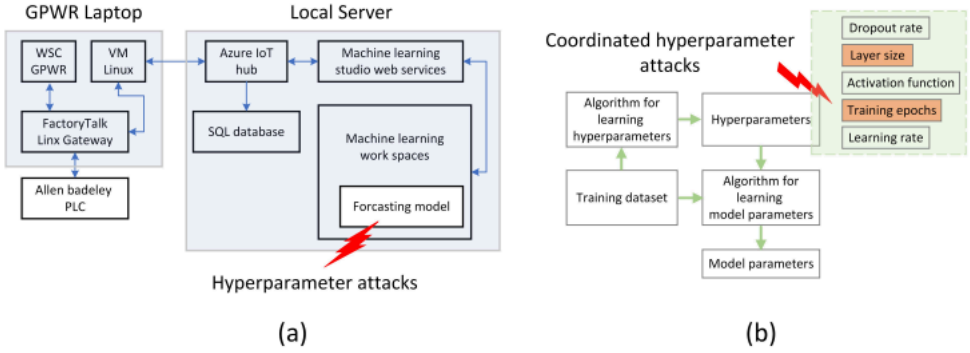


Figure 1: Hyperparameter attack on forecasting model of digital twin in nuclear power plant: (a) Digital twin; (b) coordinated hyperparameter attack on machine learning forecasting model

Dropout rate: The dropout rate ρ is a regularization technique employed in ANNs to prevent overfitting. This technique makes a portion of the incoming neurons non-functional (essentially, they're assigned a value of zero) for every update cycle while the network learns. It's expressed as

$$y'_k = y_k \times F(v_k > \rho)$$

where y_k represents the k-th entry, y'_k is the subsequent k-th output, v_k is a random value ranging from 0 to 1, and F is a representation function. If a malicious actor cleverly adjusts ρ, it can push the model towards excessive or insufficient learning, affecting its applicability and effectiveness.

Layer size: In a layer size attack, an adversary strategically modifies the size of the hidden layers in an ANN, represented as

$$H = h_1, h_2, ..., h_n \tag{1}$$

where h_1 is the size of the i-th hidden layer. The adjustment can be mathematically represented as

$$H' = h'_1, h'_2, ..., h'_n \tag{2}$$

where h'_1 represents the manipulated size of i-th layer. This deliberate distortion can undermine the ANN's ability to accurately predict outcomes by altering the model's architectural consistency.

Learning Rate: An adversary may manipulate the learning rate, denoted by η, to extremities (too high or too low) to impede the ANN's convergence. This modification is encapsulated by the equation.

$$\Delta w_{ij}(t) = \eta \cdot \delta_j \cdot x_i \tag{3}$$

where Δw_{ij}(t) is the weight change at time t, δ_j is the error term for neuron j, and x_i is the input. This malicious act can disrupt the model's gradient descent optimization process, leading to unstable weight updates, slower convergence, or even a complete failure to converge.

Number of Training Epochs: In a number of training epochs attack, the adversary manipulates the total number of epochs (iterations), denoted by E, during the model's training phase. If we denote the set of weight vectors at epoch i as W_i , then the learning algorithm tries to iteratively update Wi over E epochs to minimize a loss function L, according to

$$W_{i+1} = W_i - \eta \nabla L(W_i) \tag{4}$$

The aim is either to prematurely truncate the training process (E is set too low, i.e., E' < E), resulting in underfit models or excessively prolong it (E is set too high, i.e., E' > E), leading to overfit models. By adjusting the number of epochs, the attacker can impact the model's capacity to generalize, causing the model to be overly simplistic or overly complex. This misguidance in the learning process could lead to poor model performance and inaccurate predictions on unseen data, thus compromising the integrity of the forecasting model.

Activation Function: In an activation function attack, an adversary strategically modifies the activation function or its parameters. Mathematically, the adversary alters f(z), where z is the input to the activation function, to a manipulated function f ' (z). Activation functions introduce non-linear properties to the model, enabling the learning of complex patterns. The manipulation could be changing a sigmoid function from

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

to a hyperbolic tangent function

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{6}$$

Moreover, manipulation could be modifying parameters of a parametric ReLU (Rectified Linear Unit) function. This intentional alteration can introduce undesirable properties such as vanishing or exploding gradients that can critically impair the learning process, thereby limiting the model's ability to capture and learn complex data relationships. Consequently, the ANN's predictive accuracy and overall performance may be reduced, substantially undermining the model's forecasting capability.

In the following, attack model is advanced by encompassing two types of coordinated hyperparameter attacks based on intelligence: less intelligent and high intelligent. In a less intelligent coordinated hyperparameter attack, adversaries aim to cause significant disruptions to machine learning models, without specifically targeting the most vulnerable areas. Less advanced techniques are used to plan such attacks, typically by assigning arbitrary values to different hyperparameters. Referring to pseudo-algorithm 1, the attacker initially penetrates multiple hyperparameters, denoted as η, ρ, H, E, f(·) and then randomly assigns them inappropriate values, denoted as η ′ , ρ′ , H′ , E′ , f′ (·). If the model's accuracy diminishes, the attack is deemed successful, and the process ends with potentially leaving an opening for further exploitation. Although these random attacks lack precision, they can significantly cause considerable complications for machine learning models, disrupting their operation, and possibly leading to untrustworthy predictions. The downside of less intelligent attacks is that they can be easily detected and mitigated. For instance, one could deploy an ML-based detection system to monitor fluctuations in hyperparameters and the model's overall performance. If the detector identifies severe changes in model performance, it can infer an ongoing attack. Defenders can then deploy countermeasures, such as re-adjusting or tuning hyperparameters to their initial values or alerting system administrators. However, a less intelligent attack could unintentionally reveal the weaknesses of some hyperparameters, giving defenders important information about the model's vulnerabilities.

In contrast, a high intelligent attack is carefully designed, where the attacker conducts an complete analysis of the hyperparameters, identifying and categorizing the critical ones. Misleading values are then assigned within a certain threshold to hyperparameters η, ρ, H, E, f(·), ensuring the attack is subtle and challenging to detect. Once these changes are implemented, the attacker assesses if the model's performance has marginally declined, indicating a successful but imperceptible attack. This sophisticated attack form is designed to break past critical defenses by gradually changing the model's behavior without causing noticeable modifications that would trigger detectors to turn on. Thus, the following Algorithm 1 is constructed based on assumption 1 and assumption 2 to degrade the digital twin's ML model performance.

Assumption 1: The attacker has access to the training dataset $D_{train} = (x_i, y_i)_{i=1}^{N}$ utilized to train the machine learning model. This accessibility is pivotal for both low intelligent and high intelligent attacks, as it enables the attacker to comprehend the data distribution, make deliberate alterations to the hyperparameters η, ρ, H, E, f(·), and evaluate the model's performance against known data points. In a high intelligent attack, the attacker not only adjusts hyperparameters to η ′ , ρ′ , H′ , E′ , f′ (·) but discerns

and categorizes the critical ones, indicating an understanding of how diverse hyperparameters influence the performance measure P of the model.

Assumption: Degradation of the model's performance, as measured by performance metrics P, resulting from manipulations in hyperparameters can be discrete and gradual. That is to say, the attacker can subtly decrease the model's performance in an unnoticeable manner by judiciously selecting misleading hyperparameter values within certain thresholds. This gradual performance degradation, P ' , can be mathematically represented as P ' = P − ΔP, where ΔP is a small but statistically significant decrease in performance caused by the adjustments in the hyperparameters.

Remark 1: *The decision matrix $D_{i,j}$ in algorithm 1 was used to represent whether to attack the i-th hyperparameter at time j. The attack rate r(t) is updated at each time step according to a predefined rule. This provides the ability to adjust the intensity of the attack over time. The matrix $D_{i,j}$ stores the decision whether to attack each hyperparameter at each time step, providing a detailed record of the attack process. The functions g(t, r(t)) and h(t, r(t)) adjust the assignment of inappropriate or misleading values based on both the current time and the current attack rate.*

1.1 Digital twin system modeling with hyperparameter attack

In the realm of nuclear power plants, precise prediction of temperature variations within the reactor is of paramount importance. Accurate temperature predictions not only ensure the efficient operation of the plant but also play a pivotal role in guaranteeing safety. Over the years, there have been numerous methods employed to forecast these temperature dynamics, but the integration of machine learning (i.e., ANN), particularly using models like the Convolutional Neural Network (CNN), has emerged as a cutting-edge approach.

CNNs, traditionally employed in image and video recognition tasks, have demonstrated their versatility by being adaptable to sequence prediction tasks, such as temperature forecasting in nuclear reactors. The strength of CNNs lies in their convolutional layers, which are intrinsically designed to automatically and adaptively learn spatial hierarchies from data. When applied to sequential data from nuclear reactors, these convolutional layers effectively extract temporal features, thereby enabling the model to discern patterns and trends in temperature data over time. This innovation allows for more accurate, real-time predictions, outstripping many traditional predictive models in both efficiency and accuracy.

The digital twin, a digital representation of a physical entity or system, in this context, represents the pressurized water reactor of a nuclear power plant. By integrating the CNN-based temperature prediction model into this digital twin, reactor operators and management have access to real-time predictive insights. This integration offers a dual advantage: firstly, it allows for more proactive adjustments in the reactor's operations, ensuring optimal performance; and secondly, it serves as an early warning system, identifying potential temperature anomalies that might compromise safety. In essence, the fusion of CNNs with the digital twin technology marks a significant stride towards the modernization of nuclear reactor management, emphasizing both efficiency and safety.

Algorithm 1: Multi-Rate Time-Varying Coordinated Hyperparameter Attacks on Machine Learning Model

Output: A decision matrix $D_{i,j}$ to indicate the transition from a less intelligent attack to a high intelligent attack, and a function $r(t)$ representing the multi-rate attack pattern over time

Initialize $D_{i,j} \leftarrow$ zero matrix, $attackSuccessful \leftarrow$ false, $highIntelligenceAttack \leftarrow$ false, $t \leftarrow 0$, $r(t) \leftarrow$ initial rate ;

if $attackSuccessful = false$ **then**

  Penetrate multiple ML hyperparameters $\eta, \rho, H, E, f(\cdot)$; **while** *true* **do**

    if $highIntelligenceAttack = false$ **then**

      `// Low intelligence attack`

      Randomly assign inappropriate values $\eta', \rho', H', E', f'(\cdot)$ based on multi-rate function $g(t, r(t))$; Apply changes to the model; Evaluate performance P'; **if** $P' < P - \Delta P$ **then**

        $highIntelligenceAttack \leftarrow$ true; $D_{i,t} \leftarrow 1$ for all affected hyperparameters i; **break**;

      end

      Update the attack rate $r(t)$ based on predefined rule; $t \leftarrow t + 1$;

    end

    else if $highIntelligenceAttack = true$ **then**

      `// High intelligence attack`

      Perform analysis of hyperparameters $\eta, \rho, H, E, f(\cdot)$; Identify and classify critical ones; Assign misleading values within threshold $\eta'', \rho'', H'', E'', f''(\cdot)$ based on multi-rate function $h(t, r(t))$ and optimal stealthy approach; Apply changes to the model; Evaluate performance P''; **if** $P'' \approx P - \Delta P$ *and* $P'' < P'$ **then**

        $attackSuccessful \leftarrow$ true; $D_{i,t} \leftarrow 1$ for all affected hyperparameters i; **break**;

      end

      Update the attack rate $r(t)$ based on predefined rule; $t \leftarrow t + 1$;

    end

  end

end

return $attackSuccessful, D_{i,j}, r(t)$;

Algorithm 1: Multi-Rate Time-Varying Coordinated Hyperparameter Attacks on Machine Learning Model

The forecasting model used in digital twin is represented as

$$T(t) = \sum_{j=0}^{L_s-1} W_{d,j} \cdot \left(f_a \left(D(D_r) \cdot \underbrace{\max_{\Delta t=0}^{p-1} x(t-\Delta t) * K_j}_{\text{Convolution after MaxPooling}} \right) + b_{d,j} \right) \tag{7}$$

where T(t) - Target function at time t, L_s - Layer size, which determines the depth of the network, $W_{d,j}$ - Weight matrix for the convolution layer, f_a - Activation function, D(D_r) - Dropout function with rate D_r, K_j - Convolutional kernel for j^{th} layer, $b_{d,j}$ - Bias for the convolution layer.

In equation **Error! Reference source not found.** convolution represents the convolution operation between the input sequence x(t) and the filter K_j MaxPooling denotes the max-pooling operation applied to the convolutional output. Final Prediction gives the final temperature prediction, aggregating the convolution outputs. Given that five hyperparameters are attacked in the forecasting CNN model, their perturbations due to the attack can be represented as: δ_n, δ_ρ, δ_H, δ_E, δ_f $(\cdot)$. Then, the modified equation is:

$$T(t) = \sum_{j=0}^{H+\delta_H} \Bigg((W_{d,j} + \delta_\eta + \delta_\rho) \cdot \Bigg[\Big(f + \delta_f \Big) \Big(D(D_r + \delta_E) \cdot \max_{\Delta t=0}^{p-1} x(t - \Delta t) * (K_j + \delta_H) \Big) \Bigg]_{\text{Convolution after MaxPooling}} + (b_{d,j} + \delta_\rho) \Bigg)$$

where T(t) is the target function at time t, H and δ_H is the original number of filters and its perturbation, $W_{d,j}$, δη, and δρ are Weights and perturbations due to learning rate and momentum attacks, respectively. f(·) and δ_f (·) are activation functions. D(Dr) and δ_E are dropout function and epochs perturbation, respectively. K_j and δ_H are convolutional kernel and its perturbation, respectively. $b_{d,j}$ and δ_ρ are bias and perturbation from momentum attack, respectively.

2. Response strategy design

An adaptive predictive control framework is developed by utilizing event-triggered control law as shown in Figure 2.

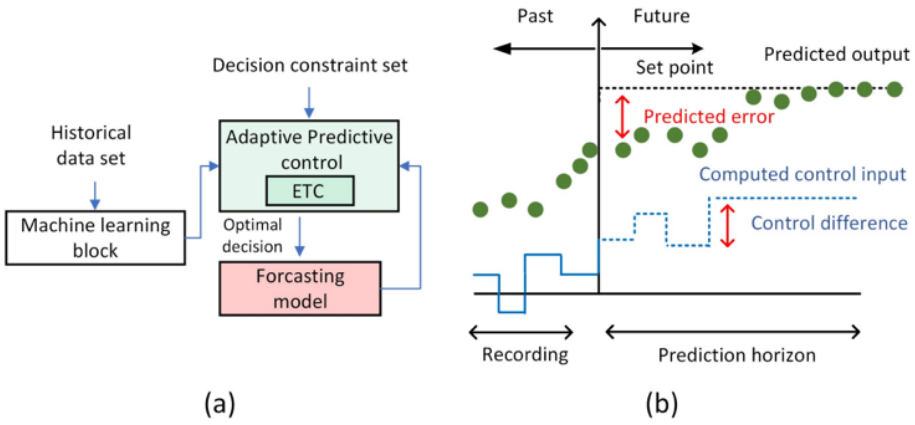


Figure 2: Response strategy design: (a) Adaptive predictive control flow with event-triggered control (ETC);

In order to develop response strategy, following control elements, such as digital twin system modeling, sliding window design, even-triggered control formulation, control action establishment, and control gain are designed.

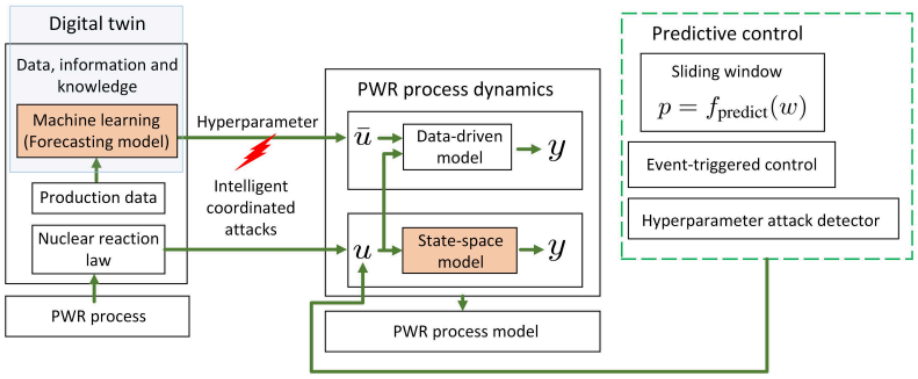
2.1 Digital twin system modeling

An attacker may exploit the sensitive dependence of the performance of learning algorithms on hyperparameters to cause the model to malfunction or produce suboptimal results. In the context of machine learning and control theory, this could mean that some of the state variables, system matrices, or controls are under the influence of hyperparameter attacks. For an adaptive control system associated with an ML model *M*, let's introduce adversarial disturbances caused due to hyperparameter attacks. Let's denote this hyperparameter attacks as δ_t which can result in two potential mappings both less intelligent and high intelligent attack:

$$\delta(t) : \{\eta, \rho, H, E, f(\cdot)\} \mapsto \{\eta^{[s(t)]}, \rho^{[s(t)]}, H^{[s(t)]}, E^{[s(t)]}, f^{[s(t)]}(\cdot)\} \tag{8}$$

where s(t) is a time-varying function that returns either ' or '' (prime or double prime) depending on the attack rate and coordination at time t. η [s(t)] is a concise way of showing that η is modified to either η ' or η '' based on the value of s(t). Equation (1) provides a compact way to represent how the attack modifies the hyperparameters over time, contingent on the value of s(t).

Figure 3: A concept of relation between control input u, state-space model forecasting model of digital twin and predictive control attack



This attack influences the system dynamics and the control law. Let's define the system dynamics by introducing the hyperparameter attack:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) + d(t) + \Delta(t)x(t) + \delta(t) \tag{9}$$

where x(t) is the state of the system, u(t) is the control input, d(t) is additive disturbances of uncertainties in the digital twin system dynamics. Δ(t) is the multiplicative uncertainties. The adversarial disturbance δ(t) could be a function of the state, control action, or some exogenous signals. Also, in adaptive control, we usually don't know the system matrices A(t) and B(t). So, we estimate them:

$$\hat{A}(t), \hat{B}(t) = f_{\text{estimate}}(A(t), B(t), x(t), u(t)) \tag{10}$$

The cost function, in this case, aims to minimize not only state and control efforts but also to minimize the effect of uncertainties. Hence, the control action becomes:

$$u(t) = -K(t)x(t) = -R(t)^{-1}\hat{B}(t)'^{T}P(t)x(t) \tag{11}$$

where K is the control gain matrix to be determined. The control gain usually changes over time in adaptive control systems to compensate for system uncertainties and adversarial disturbances. R(t) is the is a positive definite weighting matrix at time t associated with the control effort in a quadratic cost function. It quantifies the "cost" or" penalty" of using a particular control effort. Taking its inverse $-R(t^{-1})$ allows for scaling the control input. P(t) is the solution to the matrix from the Lyapunov function at time t, $\hat{B}(t)$ 'T is the transpose of the estimated system input matrix of $\hat{B}(t)$. The transpose operation allows for matrix multiplication with the state x(t) and P(t).

2.2 Sliding Window with Time-Varying Model

The sliding window retains a history of previous data points, utilizing linear regression to anticipate the subsequent data point based on the present set of data. Its size is adaptively altered in relation to the prediction error.

$$p_{\text{window}} = f_{\text{predict}}(w) \tag{12}$$

For a given data set was shown in Figure 4, the ensuing data point p_{window} is derived from: Here, $f_{predict}\ (w)$ is the regression prediction function applied to the data set w. Considering hyperparameter attacks, the window size should be adjustably changed to cater to shifting multi-rate attributes. Consequently, a time dependent variable window size, w(t), becomes necessary.

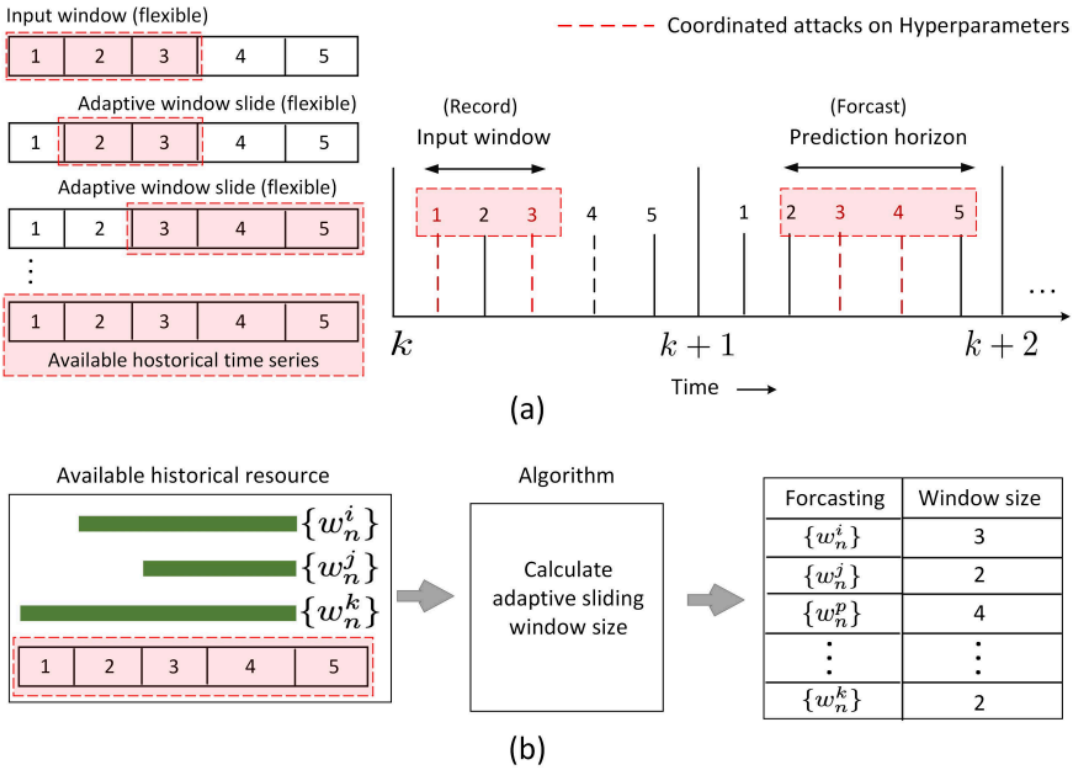


Figure 4: A concept of adaptive sliding window in predictive control: (a) Record and forecasting; (b) Calculate window size and correct hyperparameters

To introduce a concept of "flexible power," we can implement a function p(t) symbolizing this power. This function has the capability to adjust the significance or weighting of each term in the accumulation. For added intricacy, the coefficients of the auto-regressive model can be set to evolve with time:

$$p_{\text{window}} = f_{\text{predict}}(t, w(t)) = \sum_{i=t-w(t)+1}^{t} P_r(t) \cdot a_i f(x_i) \tag{13}$$

The summation's index i now spans from t−w(t)+ 1 to t, generating a dynamic window encompassing the concluding w(t) components up to the present moment t. $P_r(t)$ signifies the flexible power at the instance t, and w(t) indicates the changing window size function. This sum considers all the weighted elements a_i f(x_i) within the present window.

2.3 Event-triggered Control with Uncertainty Compensation

The event triggered control function interacts with the state-space model, data driven model, and machine learning forecasting model. A control action is only computed when the norm of the difference between the current state and the predicted state surpasses a specified threshold. Given a system state x and an event threshold ε, the control action u can also take into account the effects of uncertainties d(t) and *Δ(t*), aiming to offset them.

Let *x* symbolize the present system state, x_{pred} represent the foreseen system state, and ε stand for the event threshold. Moreover, let θ(t) be a parameter that varies with time, influencing the control action. The control action, expressed as u(t), can be described as:

$$u(t) = \begin{cases} f_{\text{control}}(x,t), & \text{if } \|x - x_{\text{pred}}\| > \varepsilon + \theta(t) \\ -K \cdot x_{\text{last}}, & \text{if } \|x - x_{\text{pred}}\| \leq \varepsilon - \theta(t) \\ u(t-1), & \text{if } \varepsilon - \theta(t) < \|x - x_{\text{pred}}\| \leq \varepsilon + \theta(t) \\ 0, & \text{otherwise} \end{cases} \tag{14}$$

with $f_{control}$*(x, t) = −K · g(x)−L· d(t)−M · Δ(t),* $f_{control}$*(x*) is the control function, computed as *−K · x* with *K* being the control gain. x_{pred} is the predicted system state. x_{last} is the last observed system state. θ(t) introduces a flexibility range around the event threshold ε, allowing for a smoother transition between the control modes.

Note that the third condition *u(t − 1)* indicates a hold condition, in which the control action persists identical to the control action at the preceding time step. This condition can be advantageous for ensuring stability or to deter abrupt alterations in the control action when the state approaches the event threshold.

Conversely, when the control action isn't activated (i.e., when | $x - x_{pred}$ | lies within the event threshold with the flexibility θ(t)), the control function is set to zero. This guarantees that the control action remains dormant when unnecessary, facilitating a seamless transition amongst control modes.

By integrating four conditions into the control action **Error! Reference source not found.** inclusive of the hold condition, we attain greater command over the system's behavior in diverse zones surrounding the event-triggering threshold. This results in enhanced efficiency and agility, especially in situations with uncertainties.

2.4 Control Action with Hyperparameter Attacks

In the training process of a neural network, a hyperparameter attack detector class is used which acts as a callback function if the validation loss exceeds a threshold, indicating a potential hyperparameter attack. It then computes a control action to correct the model's hyperparameters and applies it to the model. Given a validation loss v and a trigger threshold τ, a control action u is computed and applied to the model M as:

$$\text{if } v > \tau : \begin{cases} x & = \text{state}(v) \\ \Delta & = \text{distance}(x, x_{\text{pred}}) \\ \theta & = \text{threshold}(t) \\ u & = \begin{cases} -K \cdot g(x) - Ld(t) - M \cdot \Delta, \\ -K \cdot x_{\text{last}}, \\ u(t-1), \\ 0 \end{cases} \\ M & = f_{\text{correct}}(M, u) = M - \gamma \cdot \nabla J(M) \end{cases} \tag{15}$$

where state(*v*) transforms the validation loss *v* into a system state, $f_{control}\ (x)$ is the control function, $f_{correct}$ (*M, u*) applies the control action *u* to the model *M*.

Combining Eq. (2-8) into a single function representing the whole system, we have:

$$\text{Given variables: } x, \varepsilon, w, \tau, M \tag{16}$$

$$\text{Estimate: } \hat{A}, \hat{B} = f_{\text{estimate}}(A, B, x, u) \tag{17}$$

$$\Delta = \text{distance}(x, x_{\text{pred}}), \quad \theta = \text{threshold}(t) \tag{18}$$

The control action u is updated as

$$u = \begin{cases} -K \cdot g(x, d, \Delta), & \text{if } \Delta > \varepsilon + \theta \\ -K \cdot x_{\text{last}}, & \text{if } \Delta \leq \varepsilon - \theta \\ u(t-1), & \text{if } \varepsilon - \theta < \Delta \leq \varepsilon + \theta \\ 0, & \text{otherwise} \end{cases} \tag{19}$$

Let's consider E_{exp} as an expectation of a function f(x) over all states, where the states follow a probability distribution given by a function a_i(t) which is assumed to be normalized. Moreover, the function f(x) is controlled by the control variable u_i (t), which is governed by control rules. The correction term c_i(t) is introduced for advanced control based on the cost function J(M_i). Subsequently, we compute:

$$\begin{cases} c_i(t) = M_i - \gamma \cdot \nabla J(M_i) \\ x_i(t) = \text{state}(v_i, u_i(t), c_i(t)) \\ E_{\text{exp}} = \int f(x) a(x, t) dx \end{cases} \tag{20}$$

In this case, E_{exp} is computed as an integral, which is equivalent to a sum in continuous space. This can be thought of as the expectation of f(x) with respect to the probability distribution a(x, t) over all states x. Here a(x, t) depends on time and the state x, which is influenced by the control action ui(t) and the correction term c_i(t).

When v > τ , the system evolves as follows:

$$x = \text{state}(v), \quad u = -K \cdot g(x, 0, 0), \tag{21}$$

$$M = M - \gamma \cdot \nabla J(M), \tag{22}$$

Finally, the system outputs *M*. In **Error! Reference source not found.**, x = state(*v*), seems to be a state update equation. Here, the state x of the system is given by a function named state, which depends on the variable v. The precise meaning of this will depend on your specific system, but generally, this is

updating the state of the system based on the current variable v. Control action: The second part, $u = -K \cdot g(x, 0, 0)$, represents the control action. The control action u is computed by the negative of a gain K times a function g of the state x. The function g also takes two additional arguments that are set to zero in this case. This control law is a type of state feedback where the control action is calculated directly based on the current state x. Parameter correction: The equation **Error! Reference source not found.**, $M = M - \gamma \cdot \nabla J(M)$, represents an adjustment to the parameter(s) M. Here, M is updated by subtracting a quantity proportional to the gradient of a cost function J evaluated at M. The proportionality factor is γ, which could be seen as a learning rate. This is a standard way to perform a gradient descent update in an optimization routine, which aims to find the minimum of the cost function J with respect to the parameter(s) M.

In Algorithm 2, the two functions CalculateWindowSizePredictionHorizon() and CorrectHyperparameters() play critical roles in the overall predictive control and hyperparameter correction mechanism. They ensure the model fits the data in an optimal way while maintaining system stability, respectively.

Remark 2 *A digital twin system intricately bridges the physical entity with its digital representation. This close symbiosis, while enabling real-time monitoring and control, also introduces inherent latencies. These latencies emanate primarily from data transmission, processing, and computational delays, especially when machine learning models with varying hyperparameters are employed. Specifically, as machine learning models adapt over time—whether by hyperparameter tuning or training—the dynamics of the digital twin system can experience fluctuations. This not only challenges the determination of an optimal control gain K but also leads to treating the digital twin system as a delayed entity. To address these concerns, the Lyapunov-Krasovskii functional offers a robust mathematical framework, allowing us to analyze these delays and derive control strategies that ensure stability and resilience against dynamic disturbances in the digital twin system.*

Algorithm 2: Improved Predictive Control with Event-Triggered Adaptation

Input : System matrices A, B, and data y; hyperparameters, control parameters μ, r, a, β, λ, and κ.

Output: Updated sliding window size, corrected hyperparameters, and optimal control gain

Step 1: Initialization;
Initialize variables for LMI, define variables P, K, M, N for optimization;

Step 2: Sliding Window and Hyperparameter Correction;
Function *CalculateWindowSizePredictionHorizon()*
    Init. ialize sliding window size and resize threshold;
    Initialize time-varying multi-rate function parameters; Initialize decision matrix D;
    foreach *data point* $data$ *in the window* **do**
        Record $data$ in the sliding window;
        if *number of data points* < 2 **then**
            Continue to the next data point;
        end
        Perform linear regression data window; Predict next data point; Calculate prediction error;
        if *prediction error exceeds resize threshold* **then**
            Increase the window size;
        end
        foreach *hyperparameter* h *in hyperparameters* **do**
            Calculate error err of the system with h;
            if err *is too high* **then**
                Determine intelligence level of the attack using decision matrix D;
                if *attack is less intelligent* **then**
                    Adjust h using a lower intensity time-varying multi-rate function;
                end
                else
                    Adjust h using a higher intensity time-varying multi-rate function;
                end
            end
        end
        foreach *matrix* M *in matrices* A *and* B **do**
            Evaluate model stability with M;
            if *model is unstable* **then**
                Adjust M to improve stability;
            end
        end
    end

Step 4: Algorithm Execution;
Call CalculateWindowSizePredictionHorizon();
Call ApplyPredictiveControlAndAdaptation();

Step 5: Return Results;
Return control gain K, updated hyperparameters, and system matrices;

Algorithm 2: Improved Predictive Control with Event-Triggered Adaptation

2.5 Control Gain Design

To determine the gain K using the Lyapunov-Krasovskii method, we use the Lyapunov-Krasovskii functional (LKF) for time-delayed systems and cast the problem in a Linear Matrix Inequality (LMI)

$$V(x(t), x_\tau(t)) = x^T(t)Px(t) + \int_{-r}^{0} x_\tau^T(t+\theta)Qx_\tau(t+\theta)d\theta \tag{23}$$

where P, Q > 0 are positive definite matrices, r is the maximum delay. For stability, one aims to satisfy:

$$\dot{V}(x(t), x_\tau(t)) < 0 \tag{24}$$

Now, the following LMI (25) ensuring stability in the presence of hyperparameter attacks, based on the Lyapunov-Krasovskii functional, would be:

$$\underbrace{\begin{bmatrix} \dot{P}(t) + A(t)^T P(t) + P(t)A(t) + Q & P(t)B(t) - P(t)A(t) \\ B(t)^T P(t) & R(t) \end{bmatrix}}_{\text{System Dynamics}} + \underbrace{\begin{bmatrix} 0 & -\hat{B}(t)^T P(t) \\ -\hat{B}(t)^T P(t) & \hat{B}(t)^T \hat{B}(t) \end{bmatrix}}_{\text{Control Dynamics}} + \underbrace{\begin{bmatrix} \Sigma(t) & 0 \\ 0 & 0 \end{bmatrix}}_{\text{Hyperparameter Attack Influence}} \preceq 0. \tag{25}$$

where A is the state matrix for the non-delayed part of the system. A_r is the state matrix for the delayed part of the system. B represents control inputs. R is a weighting on the control effort. $\preceq 0$ indicates that the matrix is negative definite. $\dot{P}(t)$ arises from the fact that P(t) is time-varying in this setup. It ensures that for the given adaptive control action, the overall system remains stable. The negative definiteness of the combined matrix implies a decrease in the Lyapunov-Krasovskii functional over time.

Remark 3 *Σ(t) represents the influence of the adversarial disturbance δ(t) on the system. It models how sensitive the system is to hyperparameter attacks, with its structure and form depending on how δ(t) is incorporated into the system. To robustly counteract hyperparameter attacks, we consider incorporating robust control techniques, which might involve restructuring the control law or the system matrices.*

The resulting LMI for determining K can be represented as:

$$\begin{bmatrix} \Phi_1(t) & \Phi_2(t) \\ \Phi_3(t) & \Phi_4(t) \end{bmatrix} + \begin{bmatrix} \Sigma(t) & 0 \\ 0 & 0 \end{bmatrix} \preceq 0$$

with

$$\begin{aligned} \Phi_1(t) &\triangleq \dot{P}(t) + A(t)^T P(t) + P(t)A(t) + Q \\ \Phi_2(t) &\triangleq P(t)B(t) - P(t)A(t) \\ \Phi_3(t) &\triangleq B(t)^T P(t) \\ \Phi_4(t) &\triangleq R(t) + K(t)'^T \hat{B}(t)P(t)\hat{B}(t)^T K(t) \end{aligned}$$

where Σ(t) is a matrix that represents the adversarial disturbance effect, and P(t) is a positive definite matrix used in the Lyapunov-Krasovskii functional. R(t) is also a positive definite matrix, representing the control weighting. By solving this LMI, we can obtain the suitable P(t), R(t), and K(t), which ensures stability for the given system, even under adversarial disturbances. This LMI can be solved using specialized numerical tools, such as python, LMI solvers in MATLAB or YALMIP, to find the matrices that satisfy the inequality.

3. Implementation

Embarking on a quest to reinforce and protect digital twin models from adversarial interference in nuclear power plants, Section 3 meticulously weaves through pivotal steps and strategic methodologies, ensuring these digital entities sustain operational integrity even amidst hyperparameter attacks. Diving into the granular depths of dataset preparation and reprocessing, it underscores the imperativeness of precision and reliability in handling data that forms the bedrock upon which robust machine learning models are built. With a meticulous lens focused on managing the cascade of data from sensor-derived temperature readings within the pressurized water reactor, the initial segments prioritize establishing a clean, normalized, and adeptly preprocessed dataset, inherently recognizing its cardinal role in shaping a digital twin that mirrors the authenticity and accuracy of its physical counterpart. As the narrative transitions through testbed development, simulation setups, and exploring various nuances of hyperparameter attacks, it integrates a plethora of strategies, from minimizing control gain matrix norms to predictive control, all enveloped in a cohesive framework designed to fortify the digital twin against perturbations in its operational trajectory. The successive subsections unravel these methodologies, illustrating not only their conceptual foundations but also their practical implications and execution within a nuclear power plant scenario.

3.1 Dataset preparation and reprocessing

In the development of a digital twin forecasting model for a pressurized water reactor, the initial step revolves around dataset preparation. Leveraging historical data from sensors positioned throughout the reactor, a comprehensive dataset capturing temperature fluctuations can be compiled. Given the sensitivity and consistency required for nuclear operations, the dataset needs rigorous preprocessing to ensure accuracy. This involves the removal of outliers, normalization to bring all measurements to a common scale, and possibly smoothing to iron out minor fluctuations that don't represent genuine changes in reactor conditions. Once cleaned, this data is then shaped into appropriate input matrices suitable for a CNN, often using sliding window techniques to create sequences of temperature readings for regression forecasting.

3.2 Testbed development

A simulation setup is developed as shown in Figure 5 where CVXPY[a] (minimizes a convex objective function) is utilized to minimize Frobenius norm of the control gain matrix K, multiplied by R, a measure of control cost) subject to a linear matrix equation constraint. This linear equation constraint ensures that

[a] **CVXPY** is a Python library that helps people solve optimization problems, where you're trying to find the best solution out of many possible options, considering certain rules and limitations.

Imagine you have a piggy bank, and you want to save as much money as possible in a month. But you also must follow some rules: you can't spend more than a certain amount each day, and you need to buy both food and school supplies within the month. Trying to find the best way to save the most money, while following these rules, is like an optimization problem.

In the world of mathematics and programming, CVXPY is like a helpful tool for these kinds of problems. It allows programmers to define their problem in a very straightforward and clear manner and then solves it to find the best possible solution (like saving the most money) while still following all the rules (like daily spending limits). Specifically, it is often used for problems involving minimizing or maximizing a particular value, such as minimizing costs or maximizing profit, under certain constraints.

In a more technical sense, CVXPY is used for "convex optimization" problems.

the optimal solution aligns with the Riccati equation which arises from the Linear Quadratic Regulator (LQR) problem, a common method for finding optimal control in linear systems.

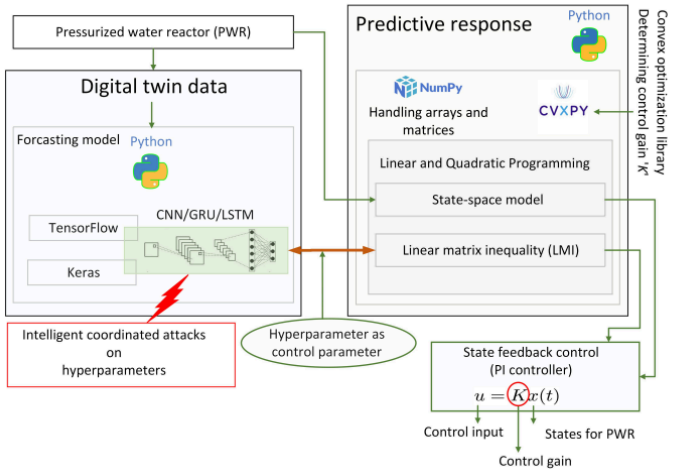


Figure 5: Testbed setup to implement proposed adaptive predictive event-triggered control for hyperparameter attacks on digital twin machine learning model

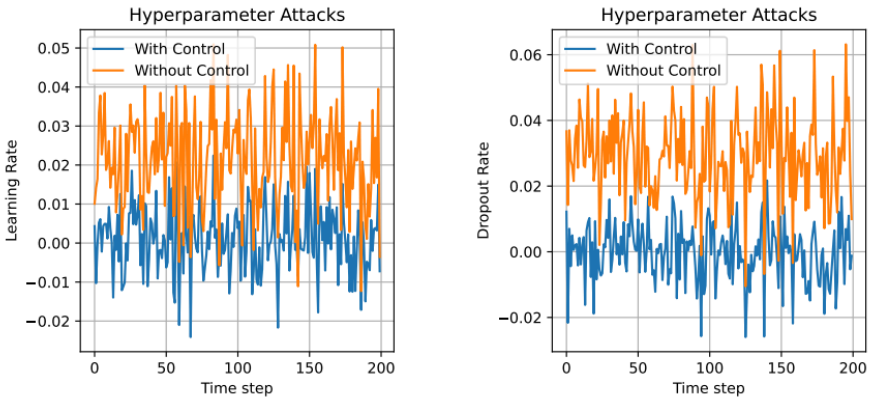
R is the cost associated with the control effort. It's used to formulate an objective function to be minimized in the controller design. A common aim is to minimize the control effort (the changes in control actions), and this is expressed as the minimization of R $*$ u, where u is the control input. This minimization needs to be balanced against the aim of keeping the system state close to a desired value, hence the use of the state cost Q.

Also, a library 'NumPy' is used for numerical computations and handling arrays and matrices A and B. In the following, function 'scipy.linalg' from SciPy is used to calculate the symmetric positive-definite solution P in the proposed secure controller. A class 'sklearn.linear model(LinearRegression)' is used to performs ordinary least squares Linear Regression from the scikit-learn library. This is used in the SlidingWindow class for predictive purposes.

3.3 Simulation

Diving into the intricate universe of machine learning models, one encounters the potential threats and disturbances instigated by hyperparameter attacks, a scenario where malicious adversaries intentionally tamper with the hyperparameters of the learning model to disrupt its predictive accuracy and reliability. Figure 6 and subsequent visuals unpack the significance and the ensuing consequences of such attacks on various aspects like learning rate and dropout rate, ultimately trickling down to affect model performance. Through a meticulous exploration of control mechanisms in the following sections, we'll explore how the deft application of control, a metaphorical steady hand on the wheel, can help navigate through the tumultuous waters of hyperparameter disturbances, ensuring model stability and integrity. Moreover, through carefully crafted visuals and analyses, we shall delve into the varying strategies employed by attackers, unearthing the impact of each on the learning model. This sets the stage not only for understanding the potential vulnerabilities and challenges posed by hyperparameter attacks but also for developing robust strategies to safeguard the model against such potential threats. Thus, we

embark on a journey of unraveling, understanding, and ultimately devising shields against the perils of hyperparameter manipulations in machine learning models.



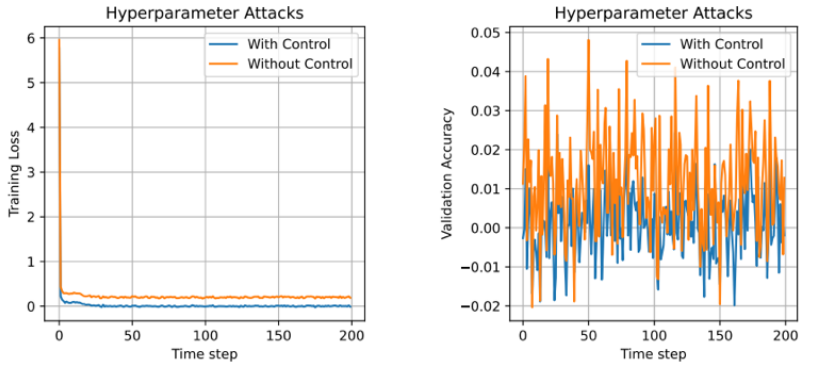
(a) Hyperparameter attacks on learning rate

(b) Hyperparameter attacks on dropout rate

Figure 6: Hyperparameter attacks on learning rate and dropout rate with control vs without control

Figure 6 (a) shows the change in learning rate under hyperparameter attacks, both with and without control. The learning rate is an essential parameter as it determines how much we adjust the weights of our network with respect to the loss gradient. When the control is applied, we observe a significant stabilization in the learning rate, despite the attack. Without control, the learning rate experiences more fluctuations, which can negatively impact the model's performance by causing the model to converge too quickly to a suboptimal solution, or by making the learning process too slow due to a very low learning rate.

Figure 6 (b) shows the impact of control on the dropout rate under hyperparameter attacks. Dropout is a technique used in neural networks to prevent overfitting. The dropout rate determines the probability at which output of a neuron is set to zero during training. From the graph, we can clearly see that applying control helps maintain the dropout rate at a more constant level. Without control, the dropout rate may become too high, leading to underfitting, or too low, leading to overfitting.



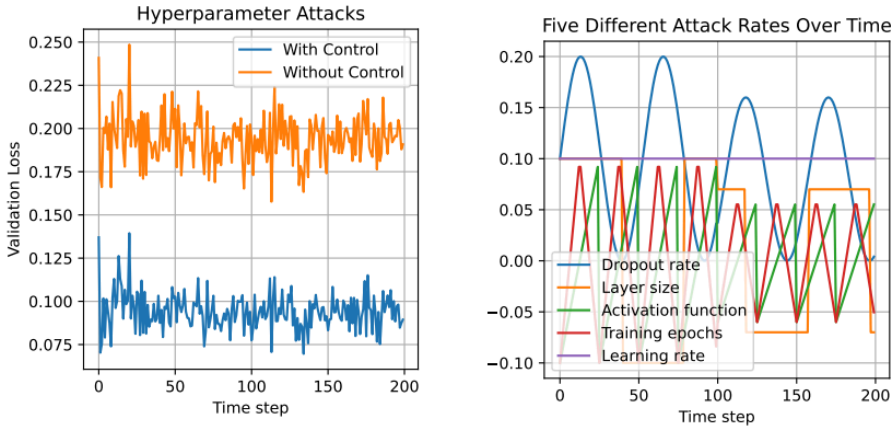
(a) Hyperparameter attacks on training loss

(b) Validation accuracy under hyperparameter attack

Figure 7: Hyperparameter attacks on training loss and validation accuracy with control vs. without control

Error! Reference source not found.(a) represents the training loss with and without control under hyperparameter attacks. Training 16 loss is an indication of how well the model is learning from the training data. From the plot, it's evident that the training loss is lower and more consistent when control is applied. Without control, the training loss shows significant fluctuations, which is undesirable as it might indicate the model isn't learning well from the training data.

Error! Reference source not found.(b) demonstrates the change in validation accuracy under attacks. Validation accuracy gives a measure of the model's performance on the validation set. The plot shows that applying control significantly improves and stabilizes the validation accuracy, despite the attacks. Without control, the validation accuracy can be significantly lower, showing the negative effect of hyperparameter attacks on the model's performance.



(a) Hyperparameter attacks on validation loss

(b) Validation accuracy under hyperparameter attack

Figure 8: Hyperparameter attacks on training loss and validation accuracy with control vs without control

Error! Reference source not found. (a) displays the validation loss under attacks. Validation loss is the value of cost function for our cross-validation data and is a measure of how well the model is doing outside the training set. When the control is applied, the validation loss is reduced and is more stable compared to the scenario without control. This indicates that the control mechanism is effectively combating the hyperparameter attacks and maintaining the generalization ability of the model. **Error! Reference source not found.** (b) shows different attack rates over a span of 200 epochs for machine learning forecasting model. Various attack profiles were implemented to analyze their effect on the forecasting model. The attack rates are manifested in five different forms:

- **Dropout rate:** This is represented by a sinusoidal attack rate, reflecting periodic alterations in the rate.
- **Layer size**: The layer size is modeled by a square wave attack rate, highlighting the abrupt changes between two levels.
- **Activation function:** It follows a sawtooth pattern for attack rate, which illustrates a linearly increasing rate followed by a sudden drop**.**
- **Training epochs:** The attack rate here follows a triangular pattern, a combination of linearly increasing and decreasing rates.

- **Learning rate:** This is maintained at a constant attack rate, serving as a baseline for comparisons.

All these rates were plotted against the attack steps, offering an insight into how different machine learning parameters can be perturbed under adversarial scenarios. The visual representation can provide a clearer understanding of the temporal dynamics of each attack type.

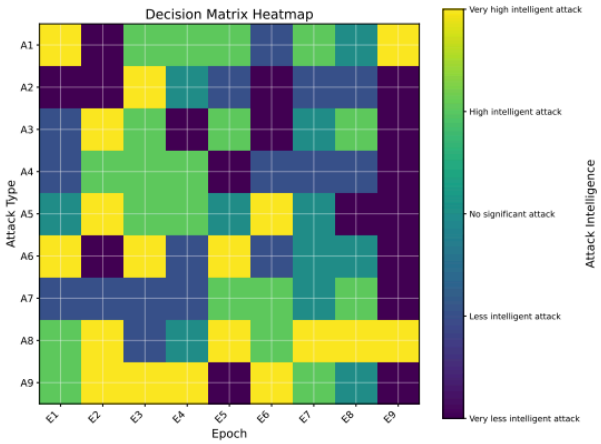


Figure 9: Decision matrix for intelligent hyperparameter attack

As shown in Figure 9, a NxN matrix is randomly generated, representing various attack types across different epochs. The matrix entries range from -2 to 2, corresponding to varying levels of attack intelligence, from very less intelligent to very high intelligent attacks. This matrix is then displayed as a heatmap, with epochs and attack types labelled along the x and y-axes, respectively.

A colorbar is added to indicate the corresponding attack intelligence levels. Gridlines are added to enhance visibility. This visualization provides an intuitive way to understand and analyze the distribution and intensity of different attacks across epochs, aiding in further mitigation strategies.

4. Offensive Use Case (Operation FrostFire)

This is a Digital Twin-Assisted Hyperparameter Attack on Autonomous Control Systems

4.1 Scenario

Thermotech Inc., an advanced thermal energy solutions company located in the bustling city of Neuropolis, has a physical security system for its main research facility. This system, as depicted in Figure 10, employs an innovative autonomous control system to monitor temperature and motion sensors throughout the facility. It alerts the security staff if any unauthorized activity is detected. Additionally, the system uses a Programmable Logic Controller (PLC) to perform control-loop actions based on sensor data.

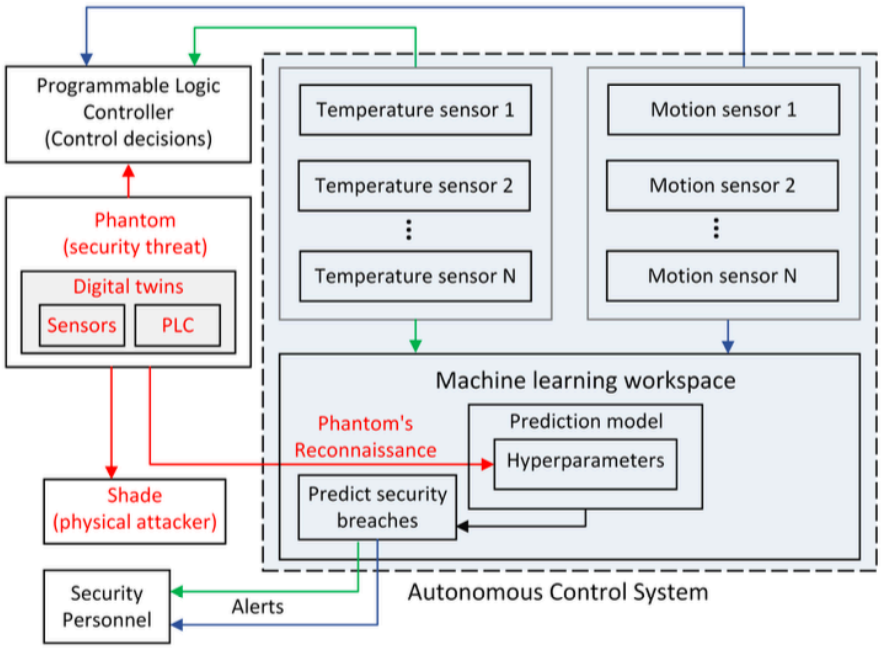


Figure 10: A Digital Twin-Assisted Hyperparameter Attack on Autonomous Control Systems

A black-hat hacker known as Phantom has targeted Thermotech Inc. with plans to breach its autonomous control system. The facility's security system heavily relies on a Machine Learning (ML) model that uses temperature and motion sensor data to predict security breaches. Under normal circumstances, human intrusion causes a sudden rise in temperature and sensor movement, thereby triggering an alert. However, the Phantom, known for exploiting ML models, has discovered this security protocol and plans to launch a hyper-parameter attack to manipulate these sensor readings. To do this, The Phantom first infiltrates Thermotech's IT infrastructure to access the data used in the ML model, thus gaining a comprehensive understanding of the regular temperature and movement data across the facility, which is crucial for the planned attack.

Parallelly, the Phantom creates two digital twins to stealthily infiltrate Thermotech Inc., a high-security research facility. One twin mirrors the facility's temperature sensors, while the other simulates the Programmable Logic Controller (PLC) responsible for control-loop actions. The hacker's plan involves subtly manipulating the temperature sensor readings over time to mimic the facility's machinery thermal signature, creating a new 'normal' for the ML model.

Also, the Phantom employs Shade for the physical intrusion. Meanwhile, the hacker tweaks the temperature readings in the digital twin environment and tests the changes in the PLC replica to avoid triggering alarms. After several weeks of gradual manipulation, the ML model perceives the infiltrator's heat sig- nature as normal, allowing Shade to access the facility undetected. This incident highlights the risks of over-reliance on ML models and digital twins without robust security measures, demonstrating that these technologies, while promising improved efficiency and capabilities, can also enable sophisticated attacks.

4.1.1 Autonomous Control System

The autonomous control system (ACS) is represented as a non-linear differential equation (1) incorporating state *(x),* temperature *(T),* motion *(M),* delay *(τ),* input *(u),* noise *(w),* and a non-linear function *f ().*

$$\begin{cases} \dot{x}(t) = & f\Big(x(t-\tau)T(t), M(t), u(t)\Big) \\ & + \int_0^t \Big(x(s) - x_{ref}(s)\Big)ds + w(t) \\ y(t) & = g\Big(x(t), u(t)\Big) + v(t) \end{cases} \tag{1}$$

where *x(t)* is the state of the system at time t, representing all the variables necessary to describe the system at a given time. *x(t − τ)* represents the state of the system at a delayed time. *T(t)* and *M(t)* are the time- varying temperature and motion inputs, respectively. u(t) is the input control signal. *w(t)* is the system noise, representing unmodelled dynamics and external disturbances. The integral term helps reduce the steady-state error by continuously correcting the deviation of *x(t)* from the desired state $x_{ref}(t)$. *f()* is a nonlinear function representing the system dynamics.

y(t) is the output of the ACS at time t. It represents the state of alarms and sensor readings. g(x(t), u(t)) is a function mapping the state x(t) and input u(t) to the output y(t). The exact form of this function would depend on the specific characteristics of the ACS. v(t) is measurement noise at time t. Real-world measurements often have some degree of uncertainty or noise. This term represents that uncertainty.

4.1.2 Sensors

We consider two sensors each for temperature and motion. The temperature and motion inputs are incorporated into the sensor equations, which now also include a sum over N different types of noise (indexed by *n=1,...,N*):

$$SoT_i(t) = g_i(Ta_i(t), T(t)) + \sum_{n=1}^{N} n_i^n(t) \quad \text{for } i = 1, 2 \tag{2}$$

$$SoM_j(t) = h_j(Ma_j(t), M(t)) + \sum_{n=1}^{N} m_j^n(t) \quad \text{for } j = 1, 2 \tag{3}$$

where $SoT_i(t)$ *and* $SoM_j(t)$ represent the temperature and motion sensor outputs, respectively. $Ta_i(t)$ and Ma_j (t) represent the actual temperature and motion readings from each sensor, which are influenced by the time-varying inputs *T(t)* and *M(t).* $n_i^n(t)$ *and* $m_j^n(t)$ represent the different types of noise for each sensor.

4.1.3 Machine Learning Model

The machine learning (ML) model includes summations over *K, L, M* dimensions, representing the dimensions of the input layer, hidden layers, and output layer, respectively:

$$H_{1k}(t) = \int_0^t dt' f(\sum_{k=1}^{K} X(t')_k W_{1k} + b_{1k}) \tag{4}$$

$$H_{2l}(t) = \int_0^t dt' g(\sum_{l=1}^{L} H_{1l}(t') W_{2l} + b_{2l}) \tag{5}$$

$$D_m(t) = \int_0^t dt' \text{softmax}(\sum_{m=1}^{M} H_{2m}(t') W_{3m} + b_{3m}) \tag{6}$$

where $H_{1k}(t)$ and $H_{2l}(t)$ represent the outputs of the first and second hidden layers of the ML model, respectively. $D_m(t)$ is the output of the decision layer of the ML model. $X(t')$, $H_{1l}(t')$, and $H_{2m}(t')$ are the inputs to the different layers at various time points t'. W_{1k}, W_{2l}, and W_{3m} are the weights of the different layers. b_{1k}, b_{2l}, and b_{3m} are the biases of the different layers. *f(), g(),* and *softmax()* are non-linear activation functions.

4.1.4 Hyperparameter Attack

To incorporate the effect of temperature sensors and motion sensors and their correlation with hyperparameters, we can consider that the hyperparameters (non-linear activation functions $f(\cdot)$, $g(\cdot)$, and $softmax(\cdot)$) are influenced to the sensor outputs $SoT_i'(t)$, and $SoM_j'(t)$.

First, we have the manipulated sensor outputs that are affected hyperparameter attacks over time:

$$SoT_i'(t) = SoT_i(t) + \int_0^t dt' \delta_i(t') \quad \text{for } i = 1, 2 \tag{7}$$

$$SoM_j'(t) = SoM_j(t) + \int_0^t dt' \epsilon_j(t') \quad \text{for } j = 1, 2 \tag{8}$$

where $SoT_i'(t)$ and $SoM_j'(t)$ are the manipulated sensor outputs, and $\delta_i(t')$ *and* $\varepsilon_j(t')$ represent the attacker's hyperparameter manipulations at various time points t'.

Now, let's assume that these manipulated sensor outputs are directly influenced by the rate of change of the manipulated hyperparameters. Therefore, the rate of change of the hyperparameters can be modified as follows:

$$\frac{df'}{dt} = r(t) f' + \int_0^t \Omega(f', r(t), SoT', SoM') dt \tag{9}$$

$$\frac{dg'}{dt} = r(t) g' + \int_0^t \Omega(g', r(t), SoT', SoM') dt \tag{10}$$

$$\frac{d(\text{softmax})'}{dt} = r(t)(\text{softmax})' + \int_0^t \Omega((\text{softmax})', r(t), SoT', SoM') dt \tag{11}$$

where $f'(\cdot)$, $g'(\cdot)$ and $(softmax)'(\cdot)$ are manipulated non-linear activation function. Attack rate $r(t)$ at time t changes over time according to a predefined rule. $\Omega(., r(t))$ is a function that defines the multi- rate attack pattern based on the manipulated hyperparameter and the current attack rate. After that, these hyperparameters will then be used in the computation of the manipulated decision of the model, D_m':

$$H'1k(t) = \int 0^t dt' f'(\sum_{k=1}^{K} X'(t')kW1k + b_{1k}) \tag{12}$$

$$H'2l(t) = \int 0^t dt' g'(\sum_{l=1}^{L} H'1l(t')W2l + b_{2l}) \tag{13}$$

$$D'm(t) = \int 0^t dt' \text{softmax}(\sum_{m=1}^{M} H'2m(t')W3m + b_{3m}) \tag{14}$$

where H'1k(t), H'2l(t), and Dm' (t) represent the outputs of the manipulated ML model. The correlation between the hyperparameters and the sensor readings is modeled by the function Ω, which depends on the current hyperparameters, the current sensor readings, and the attack rate. This approach provides a mathematical way to capture the relationship between the sensors, hyperparameters, and the attack's evolution.

Let the sensor readings $SoT_i(t)$, $SoM_j(t)$ are inputs to the system that influence decision matrix Di,j, which determine the label of attack intelligence. The manipulated sensor readings, $SoT_i'(t)$, and $SoM_j'(t)$ are derived from $D_{i,j}$ and have an effect on $D'_m(t)$. We also assume that the hyperparameters are manipulated based on $D_{i,j}$ and have an influence on $D'_m(t)$. Let $H(D_{i,j}, SoT_i(t), SoM_j(t))$ be a transformation function that takes the decisions tracked by $D_{i,j}$ and the sensor readings $SoT_i(t)$, and $SoM_j(t)$ as inputs and produces the manipulated sensor readings and hyperparameters as outputs. This transformation can be represented as:

$$\left[\ SoT_i'(t)\ SoM'j(t)\ f'(\cdot)\ g'(\cdot)\ (\text{softmax})'(\cdot)\ \right] = \mathcal{H}(Di,j,SoT_i(t),SoM_j(t)) \tag{15}$$

The manipulated decision $D'_m(t)$ can be derived from these manipulated sensor readings and hyperparameters. This relationship can be written as:

$$D'_m(t) = \mathcal{G}(SoT_i'(t), SoM_j'(t), f'(\cdot)\ g'(\cdot)\ (\text{softmax})'(\cdot)) \tag{16}$$

Where G(·) is a function that takes the manipulated sensor readings and hyperparameters as input and computes the manipulated decision $D'_m(t)$.

Please note that $H(\cdot)$ and $G(\cdot)$ are complex functions that model the process of the phantom attack and the manipulated decision making. They should be defined based on the specific dynamics of the system. Also, the indexes i and j in $D_{i,j}$, $SoT_i'(t)$, and $SoM_j'(t)$ need to be handled properly to reflect the temporal and spatial changes of the system.

The decision matrix $D_{i,j}$ can be mathematically described as follows. Let ΔP be the difference in performance defined as $\Delta P = P - P'$, where P is the performance of the model at current time and P' is the performance of the model at the previous timestep. Also, let's assume δ_P is a performance threshold defined in the model. Now, we can define $D_{i,j}$ as:

$$D_{i,j} = \begin{cases} 2 & \text{if } \Delta P < -\delta_P\ 1 \\ \text{if } -\delta_P \le \Delta P < -T_1\ 0 & \text{if } -T_1 \le \Delta P < T_1\ -1 \\ \text{if } T_1 \le \Delta P < T_2\ -2 & \text{if } \Delta P \ge T_2 \end{cases} \tag{17}$$

where $D_{i,j}$ = 2: represents very high intelligent attack. $D_{i,j}$ = 1: represents high intelligent attack. $D_{i,j}$ = 0: represents no significant attack. $D_{i,j}$ = −1: represents less intelligent attack. $D_{i,j}$ = −2: represents very less intelligent attack. T_1 and T_2 are the defined thresholds, with T_1 = 0.1 and T_2 = 0.2. Here i and j in $D_{i,j}$ represent the corresponding time step and the particular agent respectively.

In what follows, to incorporate the use of temperature and motion sensors into the attack algorithm (see 4.2 Simulation), new variables S_{ti} and S_{mj} are included to represent the current state of the temperature and motion sensors respectively. A predefined function CheckForStealth *(t, S_{ti}, S_{mj})* is considered that analyzes the current sensor readings and determines whether the conditions are suitable for a stealthy attack. This function returns true if the conditions are met and false otherwise.

4.2 Simulation

In this experiment, the function apply control in Figure 11 takes an input vector u that dictates how each hyperparameter can be compromised. The method modifies the activation function of each layer. This function ensures that the new hyperparameters are within reasonable bounds. The manipulation of these hyperparameters signifies an attack on the machine learning model by maliciously altering its learning and prediction patterns.

```
123
124  def apply_control(self, u):
125      new_lr = max(0.001, self.model.optimizer.lr.numpy() + u[0])
126      self.model.optimizer.lr.assign(new_lr)
127
128      for layer in self.model.layers:
129          if isinstance(layer, tf.keras.layers.Dense):
130              current_activation = layer.get_config()["activation"]
131
132              # Attack on activation function
133              if current_activation == 'relu': # f
134                  layer.activation = tf.keras.activations.sigmoid  # changing to g
135              elif current_activation == 'sigmoid': # g
136                  layer.activation = tf.keras.activations.softmax  # changing to softmax
137              else:  # softmax or any other activation
138                  layer.activation = tf.keras.activations.relu  # changing to f
139
```

Figure 11: Hyperparameter attack (compromised activation functions)

The decision matrix serves as a blueprint for the attacks as shown in Figure 12, assigning different attack levels based on the differential performance of a model. The function update decision matrix is designed to classify attacks based on their intelligence level according to their effect on the model's performance. The intelligence of the attack is gauged using performance metrics, with the assumption that an attack is more intelligent if it can cause a more significant drop in the performance of the model. The update decision matrix function updates the decision matrix based on the performance difference of the model. The intelligence of the attack is ranked from -2 (very less intelligent) to 2 (very high intelligent), depending on the severity of the performance drop. The function takes as input the index (i), the prime performance of the model performance prime, and the model itself.

```
246  # Decision matrix
247
248  def update_decision_matrix(self, i, performance_prime, model):
249      """Update the decision matrix based on performance."""
250      thresholds = [0.2, 0.1]  # Define your thresholds
251
252      performance_diff = model.performance - performance_prime
253
254      if performance_diff < -model.delta_P:
255          self.D[i, self.t] = 2  # Very high intelligent attack
256      elif performance_diff < -thresholds[0]:
257          self.D[i, self.t] = 1  # High intelligent attack
258      elif performance_diff < thresholds[0]:
259          self.D[i, self.t] = 0  # No significant attack
260      elif performance_diff < thresholds[1]:
261          self.D[i, self.t] = -1  # Less intelligent attack
262      else:
263          self.D[i, self.t] = -2  # Very less intelligent attack
264
```

Figure 12: Decision update in hyperparameter attacks

The thresholds for determining the level of intelligence are set as [0.2, 0.1]. If the performance drop is below the negative of the model's δP, it is considered a very high intelligent attack and assigned a rank of 2 in the decision matrix. If it is below -0.2, it is a high intelligent attack.

Algorithm 1: Hyperparameter Attacks on Machine Learning Model with Sensors

Output: A decision matrix $D_{i,j}$ to indicate the transition from a less intelligent attack to a high intelligent attack, and a function $r(t)$ representing the multi-rate attack pattern over time

Initialize $D_{i,j} \leftarrow$ zero matrix, $attackSuccessful \leftarrow$ false, $highIntelligenceAttack \leftarrow$ false, $t \leftarrow 0$, $r(t) \leftarrow$ initial rate ;

if $attackSuccessful = false$ **then**

  Penetrate multiple ML hyperparameters $f(\cdot)$, $g(\cdot)$, and softmax$(\cdot)$); **while** *true* **do**

    Read sensor data S_{t_i} and S_{m_j} for all i and j; **if** $highIntelligenceAttack = false$ **then**

      `// Low intelligence attack`

      Randomly assign inappropriate values $f'(\cdot)$, $g'(\cdot)$, and softmax$'(\cdot)$) based on multi-rate function $g(t, r(t))$; Apply changes to the model; Evaluate performance P'; **if** $P' < P - \Delta P$ *and CheckForStealth(t,* S_{t_i}, S_{m_j}*) = true* **then**

        $highIntelligenceAttack \leftarrow$ true; $D_{i,t} \leftarrow 1$ for all affected hyperparameters i; **break**;

      end

      Update the attack rate $r(t)$ based on predefined rule; $t \leftarrow t + 1$;

    end

    else if $highIntelligenceAttack = true$ **then**

      `// High intelligence attack`

      Perform analysis of hyperparameters $f(\cdot)$, $g(\cdot)$, and softmax$(\cdot)$); Identify and classify critical ones; Assign misleading values within threshold $f'(\cdot)$, $g'(\cdot)$, and softmax$'(\cdot)$) based on multi-rate function $h(t, r(t))$ and optimal stealthy approach; Apply changes to the model; Evaluate performance P''; **if** $P'' \approx P - \Delta P$ *and* $P'' < P'$ *and CheckForStealth(t,* S_{t_i}, S_{m_j}*) = true* **then**

        $attackSuccessful \leftarrow$ true; $D_{i,t} \leftarrow 1$ for all affected hyperparameters i; **break**;

      end

      Update the attack rate $r(t)$ based on predefined rule; $t \leftarrow t + 1$;

    end

  end

end

return $attackSuccessful, D_{i,j}, r(t)$;

To instantiate a decision matrix, we can create an empty matrix of the same size as the maximum number of iterations or time points you expect in your program. For example, if we want a m by n matrix, we can initialize it like this:

$$\text{self}.D_{i,j} = np.\text{zeros}((m, n)) \tag{18}$$

In this way, a m by n decision matrix is created with all entries initially set to zero. As shown in **Error! Reference source not found.**, a $m \times n$ matrix is randomly generated, representing various attack types across different epochs. The matrix entries range from -2 to 2, corresponding to varying levels of attack intelligence, from very less intelligent to very high intelligent attacks. This matrix is then displayed as a heatmap, with epochs and attack types labelled along the x and y-axes, respectively.

A colorbar is added to indicate the corresponding attack intelligence levels. Gridlines are added to enhance visibility. This visualization provides an intuitive way to understand and analyze the distribution and intensity of different attacks across epochs, aiding in further mitigation strategies.

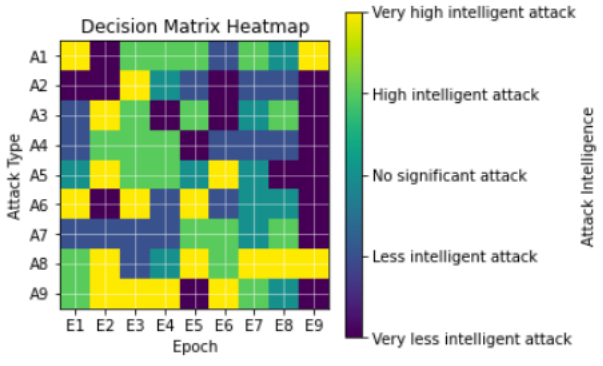


Figure 13: Decision matrix heatmap for intelligent hyperparameter attack

5. Defensive Use Case (Operation MirrorShield)

This is a Proactive Defense against Hyperparameter Attack using Digital Twins

5.1 Scenario

In the city of Cyberscape, nestled among its skyscrapers, sits a prominent financial institution, TrustBank. This entity deals with highly confidential data and vast sums of money, making its security a matter of paramount importance. TrustBank employs a state-of-the-art security system using an autonomous control system to regulate and monitor access to its data centers, incorporating advanced temperature and motion sensors.

In the shadowy realms of the cyber underworld, an entity known as The Riddler plans to break into TrustBank's data centers. The Riddler is known for their proficiency in exploiting Machine Learning (ML) models and is aware of the autonomous control system that TrustBank uses. However, TrustBank's cyber-security team, led by the formidable Falcon, is always one step ahead.

Falcon and her team use an ML model to predict potential security breaches using temperature and motion sensor data. Aware of the vulnerabilities, Falcon's team uses a combination of live data and Digital Twins to constantly monitor for anomalies.

They have a digital replica of both the temperature sensor and the PLC running the control loops, mirroring the actual operations in real-time. This parallel system acts as a playground to simulate attacks and understand the behavior of the ML model under various threat scenarios without exposing the actual control system to risks.

One day, the system flags a slight discrepancy between the behavior of the ML model in the actual system and its Digital Twin. The model in the real world has started accepting a slightly higher temperature as normal compared to its twin. Falcon instantly recognizes this as a potential hyperparameter attack.

The Riddler, who had managed to infiltrate the TrustBank's IT infrastructure, had started subtly manipulating the temperature sensor data in an attempt to deceive the ML model. He intended to trick the model into accepting higher temperatures, paired with no motion, as normal, paving the way for a physical intruder to bypass the heat sensor.

However, Falcon's team was prepared. Noticing the deviation in the behavior of the real model and its digital twin, they initiated an investigation. They employed differential analysis, comparing the hyperparameters and the learning patterns of both the twins. The discrepancies confirmed Falcon's suspicion of a hyperparameter attack.

Falcon acted swiftly, cutting off the access to the intruder, resetting the model to a safe checkpoint before the attack, and retraining it with verified data. Simultaneously, TrustBank increased its physical security measures, making sure any physical intrusion attempts would be thwarted.

The Riddler's attempts were stymied, his plans foiled. TrustBank remained secure, thanks to Falcon's innovative defensive strategy of using Digital Twins as a constant audit mechanism for their ML model.

This event underscores the importance of proactive measures and continuous monitoring in cyber defense, illuminating how Digital Twins can serve as a powerful tool to detect and counter sophisticated attacks on ML models.

5.2 System modeling for defense strategy

System modeling with predictive control approach considers the autonomous control system, sensors, ma- chine learning model, and hyperparameter attack. These components interact in complex ways, and their relationships must be captured in the predictive model. Given that, the linear regression model to predict the possibility of a hyperparameter attack in the next step can be enhanced by considering these components.

Incorporating Autonomous Control System (ACS) State: The state of the ACS, *x(t),* can have an influence on the likelihood of an attack.

Incorporating Sensor Data: The outputs of the temperature and motion sensors, *$SoT_i(t)$ and $SoM_j(t)$,* give us direct readings of the environment, which might be relevant.

Incorporating ML Model Decision Layer Output: The output of the decision layer of the ML model, Dm(t), can provide insights into the current decision-making process of the model.

Now we can predict the possibility of a hyperparameter attack as follows:

$$\hat{y}_i = \beta_0 + \sum_{j=1}^{m} \beta_j D_{i,j} + \gamma_1 x(t) + \gamma_2 \sum_{i=1}^{2} SoT_i(t) + \gamma_3 \sum_{j=1}^{2} SoM_j(t) + \gamma_4 D_m(t) + \epsilon_i$$

where $\hat{y}_i$ represents the predicted possibility of a hyperparameter attack for the i^{th} step. β_j are the coefficients for each past state j. γ_1, γ_2, γ_3, and γ_4 are coefficients for the ACS state, temperature sensor outputs, motion sensor outputs, and ML model's decision layer output, respectively. ε_i is the error term.

5.3 Defense for decision matrix of hyperparameter attack

To deal with the decision matrix of hyperparameter attacks, we can integrate a predictive control approach to take corrective action based on predicted future behavior of the system. The decision matrix for hyperparameter attack detection would be a binary matrix D of size $n \times m$, where n is the number of hyperparameters, and m is the number of previous system states observed. Each element $D_{i,j}$ in the matrix would represent whether a hyperparameter attack was detected (1) or not (0) for the i^{th} hyperparameter at the j^{th} previous state.

Remark 1 *In the Data-Driven Iterative Learning Predictive Control (DDILPC) approach, the predictive control system could then take action based on these predictions: If the predicted probability of a hyperparameter attack for a given parameter exceeds a threshold θ, the system could initiate the defensive strategy. The system could also take preventive measures, such as changing the learning rate or other hyperparameters, to make it harder for the attack to succeed. The system could log the attack for future reference and analysis. This predictive control approach gives the DDILPC system the ability to anticipate and respond to potential hyperparameter attacks before they occur. The decision matrix serves as the historical data to train the predictive model and make informed decisions on future actions.*

Integrating a predictive control approach to handle the decision matrix for hyperparameter attack detection, the mathematical formulation can be presented as follows:

We start by defining the decision matrix $\mathrm{D} \in \mathrm{R}^{\mathrm{n}\times\mathrm{m}}$, where n is the number of hyperparameters, and m is the number of observed past states of the system. Each entry a_{ij} indicates whether an attack has been detected (1) or not (0) for the i^{th} hyperparameter at the j^{th} previous state.

In the DDILPC approach, we then take appropriate action based on these predictions. This can be mathematically described as a decision function f :

Given the decision matrix $D_{i,j}$ that characterizes the changes in the prediction ΔP, we can revise the function $f(\hat{y}_i,\theta)$ to take into account these specific conditions on ΔP.

$$f(\hat{y}i, Di, j) = \begin{cases} \text{initiate high-level defense strategy} & \text{if } \hat{y}i > \theta \text{ and } Di, j = 2 \\ \text{initiate medium-level defense strategy} & \text{if } \hat{y}i > \theta \text{ and } Di, j = 1 \\ \text{maintain status quo} & \text{if } D_{i,j} = 0 \\ \text{update learning rate moderately} & \text{if } \hat{y}i \leq \theta \text{ and } Di, j = -1 \\ \text{update learning rate significantly} & \text{if } \hat{y}i \leq \theta \text{ and } Di, j = -2 \end{cases}$$

The function f now integrates the decision matrix $D_{i,j}$ with the hyperparameter attack predictions $\hat{y}_i$, making it more sophisticated and fine-grained in terms of initiating defense strategies and updating learning rates.

Specifically, the system initiates a high-level defense strategy when a high probability of attack ($\hat{\mathrm{y}}\mathrm{i} > \theta$) coincides with a significant decrease in prediction ($D_{i,j} = 2$). On the other hand, the learning rate update is made more flexible, with larger updates ($D_{i,j} = -2$) when the probability of attack is low but the prediction has significantly increased. In this equation, if the predicted probability of a hyperparameter attack $\hat{\mathrm{y}}_\mathrm{i}$ for a given hyperparameter exceeds a certain threshold θ, the system initiates a defense strategy. Otherwise, the system could take preventive measures by adjusting the learning rate or other hyperparameters, making it harder for the attack to succeed.

Therefore, the DDILPC system is equipped with the ability to anticipate and respond to potential hyper- parameter attacks based on predictions from historical data. This results in an intelligent and robust control system that can handle potential adversarial attacks effectively.

5.4 Data-Driven Iterative Learning Predictive Control (DDILPC)

Iterative Learning Control (ILC) is a control method that uses historical data from the previous iteration to improve control performance for the next iteration. On the other hand, Predictive Control uses the model of the system to predict future outputs and adjust the control variables accordingly.

When we combine the two into a DDILPC approach, we get a control method that uses the historical data to improve prediction accuracy and control performance over iterations. It makes a trade-off between data-driven learning and model-based prediction.

1. **FeedForwardNetwork Initialization:**
   Initialize FeedForwardNetwork object with random weights W_1, b_1, W_2, b_2, W_3, b_3.

$$W_1 \in R^{5\times5}, \quad b_1 \in R^5$$
$$W_2 \in R^{5\times2}, \quad b_2 \in R^2$$
$$W_3 \in R^{2\times1}, \quad b_3 \in R^1$$

2. **Decision Matrix Initialization:**
   Initialize the Decision Matrix object to capture historical model decisions and provide a reference framework to identify anomalies or potential attacks.

   Represent DecisionMatrix by M with dimensions $M \in R^{t \times d}$, where *t* captures historical data and *d* logs various decision parameters.

3. **Trust Bank Control System Initialization:**
   Incorporate FeedForwardNetwork into the TrustBankControlSystem as a digital twin, enabling real-time validation against the primary model.

   If the primary model is denoted $F(\theta)$, the digital twin is represented as $F'(\theta')$.
   Synchronize θ' with θ at regular intervals.

4. **Control System and Predictive Steps:**
   Set up the ControlSystem to oversee the predictive steps and manage hyperparameter adjustments in the FeedForwardNetwork dynamically.

   Let N be the count of predictive steps, and define C as the controls at each interval: C = {c1, c2, ..., cN }.

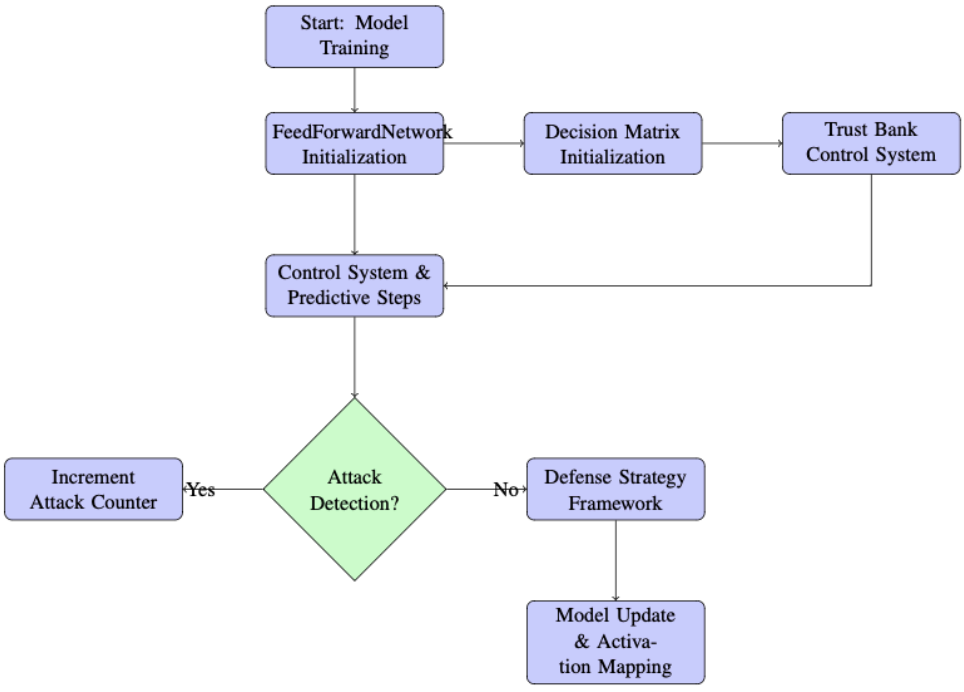


Figure 14: Data-Driven Iterative Learning Predictive Control (DDILPC) strategy. Integrates ILC and Predictive Control concepts, employing a FeedForwardNetwork, Decision Matrix, and Trust Bank Control System ensuring optimal performance while countering threats.

5. **Attack Counter and Defense Strategy Framework:**

We're setting up a protective system against potential tampering with machine learning model settings, known as hyperparameter attacks. We begin by introducing a counter, C_{attack}, which acts as a vigilant sentinel, constantly monitoring for suspicious activities. A threshold, θ, acts as our alarm level—if the counter crosses this threshold, it's a sign that something's off. If this alarm sounds, we spring into action: we return our model settings to safe values, notify our tech team of the potential threat, and might even temporarily stop our model from further training to ensure it's not unknowingly learning from harmful data. Lastly, just as tech keeps evolving, so does our protective system. We continuously revise our defense measures to remain ahead of new threats and challenges.

Hyperparameter Attack Counter Initialization: Introduce a counter C_{attack} to quantitatively monitor anomalies that suggest possible hyperparameter attacks. This counter is integral to the defense strategy, serving as an early warning system. Mathematically, initialize:

$$C_{attack} := 0$$

Threshold Determination: Define a threshold θ based on domain expertise, historical data, or heuristics. When C_{attack} exceeds θ, it indicates a probable attack or an abnormal behavior in the system, triggering the defense mechanism:

$$\text{Trigger Defense} \Longleftrightarrow C_{attack} > \theta$$

Formulating the Defense Strategy: The defense mechanism is multifaceted, and its actions depend on the severity and nature of the detected anomaly. Potential actions include:

(a) **Hyperparameter Restoration:** Reset hyperparameters to their last known secure settings or to pre-defined safe values. If θ_h represents a set of trusted hyperparameters, then upon detection of an anomaly:

$$\theta_{current} \leftarrow \theta_h$$

(b) **Alert System Administrators:** Inform system administrators or relevant stakeholders about the potential threat. This could be done through various channels, be it email, system logs, or real-time notifications.

(c) **Model Training Pause:** Temporarily halt model training to prevent potential propagation of erroneous updates or malicious modifications. Formally, if T represents the training process:

$$T\ (\theta_{current,}\ \text{data}) \rightarrow \text{Pause if } C_{attack} > \theta$$

Continuous Monitoring and Adjustment: Regularly review and adjust the threshold θ and defense strategies based on new insights, evolving threat models, and system requirements to ensure robust protection.

6. **Model Training:**

In this section, we dive into the continual training of a model while keeping a keen eye on safeguarding it. Initially, we set up an input matrix and calculate intermediate layers using specific mathematical functions. Then, we determine the output and subsequent predictions with some additional calculations, involving a method known as 'softmax'. As we step through the predictive process, we vigilantly check any changes in the learning rate against a specific criterion and modify it by referring to a DecisionMatrix, current predictions, and previous controls. DecisionMatrix is then updated with new data about the learning rate and predictions. However, it's not just about training — we're on guard for any unusual behaviors or sudden drops in performance. If such discrepancies are noted, our counter, C_{attack}, is incremented. And if C_{attack} crosses a certain changing threshold, θ(t), our defense mechanisms are activated, ensuring our model remains secure and trustworthy amidst its learning journey.

Process includes:

a. Creating an input matrix $X \in R^{n \times d}$.

b. Calculating intermediate layers H_1 and H_2:

$$H_1 = \tanh(XW_1 + b_1)$$

$$H_2 = \tanh(H_1W_2 + b_2)$$

c. Deriving the softmax output D and resultant predictions:

$$D = \exp(H_2W_3 + b_3)$$

$$Predictions = \frac{D}{\sum D}$$

d. For each predictive step:

   i. Validate any learning rate alterations against a pre-defined criterion.

   ii. Modify the learning rate Œ∑ based on DecisionMatrix, current model prediction, and past controls.

   iii. Refresh DecisionMatrix with updated learning rate and prediction data.

   iv. Monitor model performance. On detecting significant performance drops or irregular learning rate fluctuations, increase C_{attack}.

   v. Activate the defense strategy if C_{attack} surpasses a dynamic threshold $\theta(t)$.

7. **Model Update and Activation Function Mapping:**

Mapping Strategy: Establish a function M that maps from a set of defined activation functions A to a set of hyperparameters H. This function assists in dynamically choosing the optimal activation function and corresponding hyperparameters based on the model's requirements.

$$M: A \rightarrow H \; a \; 7 \rightarrow h$$

$$\theta_{current} \leftarrow \theta_h$$

where a is an activation function from A and h is the corresponding hyperparameters in H.

Activation Functions: Consider a suite of activation functions such as Sigmoid σ(x), Tanh tanh(x), and ReLU max(0, x) as members of A.

Dynamic Model Update: When a model update is needed, use the mapping function M to determine the most suitable activation function and corresponding hyperparameters. Update the model's architecture and parameters accordingly. Let F(X;θ,a) denote the model with parameters θ and activation function a. On update,

$$F\ (X;\ \theta,\ a_{old}) \rightarrow F\ (X;\ \theta^{'},\ M(a_{new}))$$

where $\theta^{'}$ are the updated parameters and a_{new} is the newly selected activation function.

Learning Rate Adjustment: During this model update process, the learning rate η might also need adjustments. If Δη represents the change in learning rate due to the new activation function and other factors, the updated learning rate becomes:

$$\eta_{updated} = \eta + \Delta\eta$$

Algorithm 1: Data-Driven Iterative Learning Predictive Control with Defense against Hyperparameter Attacks

Result: Predictive control for model with learning rate adjustment and defense mechanism against hyperparameter attacks

Initialize FeedForwardNetwork object (with random weights $W_1 \in \mathbb{R}^{5\times5}$, $b_1 \in \mathbb{R}^5$, $W_2 \in \mathbb{R}^{5\times2}$, $b_2 \in \mathbb{R}^2$, $W_3 \in \mathbb{R}^{2\times1}$, $b_3 \in \mathbb{R}$); **Initialize** TrustBankControlSystem object with FeedForwardNetwork as digital twin; **Initialize** ControlSystem object with predictive steps; **Initialize** counter for hyperparameter attacks and a defense strategy;

while *model training is not finished* **do**

  Define input array $X \in \mathbb{R}^{n\times d}$; **Compute** $H_1 = \tanh(XW_1 + b_1)$; **Compute** $H_2 = \tanh(H_1W_2 + b_2)$; **Compute** $D = \exp(H_2W_3 + b_3)$; Predictions = $D/\sum D$; **for** *step in range(predictive steps)* **do**

    prediction = Predictions[step]; **if** *control history is not empty* **then**

      last control = control history[-1]; ΔP = prediction - last control; Define $D_{i,j}$ according to ΔP; Evaluate $f(\hat{y}i, Di, j)$ and take the corresponding action;

    end

    append prediction to control history; **if** *length of control history* > 1000 **then**

      remove the first element from control history;

    end

    if *significantly reduced model performance or suspiciously high fluctuation in learning rate* **then**

      increment counter for hyperparameter attacks; **if** *counter for hyperparameter attacks exceeds certain threshold* **then**

        activate defense strategy;

      end

    end

  end

end

Define activation mapping for activation function update (relu to sigmoid, sigmoid to softmax);

Call update model function with learning rate delta and activation mapping;

5.5 Simulation

Section 5.5 delineates a thorough simulation of defensive capabilities within machine learning model control systems, utilizing the "ControlSystem" class and subsequent functions, designed to safeguard model functionality and preserve data integrity against hyperparameter attacks. Commencing with a baseline scenario, sensor data - characterized by "Real" and "Twin" sensors - undergoes an examination, with both demonstrating nominal, synchronized readings. Subsequently, a hyperparameter attack is introduced, inducing substantial, erratic fluctuations in the "Real" sensor, leading to a divergent data trajectory when juxtaposed with the "Twin" sensor. Following this aberration, the implementation of a defensive mechanism is examined, applying a dampening strategy, specifically a simple moving average, to the perturbed "Real" sensor data. The resultant effect of this strategy is explored through graphical analysis, revealing a subsequent convergence of the "Real" and "Twin" sensor data post-attack. Additionally, an anomaly detection methodology is deployed, identifying and annotating data discrepancies between the "Real" and "Twin" sensors that exceed a predefined threshold. This section provides an in-depth exploration of the defensive mechanism's functionality and efficacy in mitigating hyperparameter attack effects and identifying potential anomalies within the simulation timeframe.

5.5.1 ControlSystem Class Overview:

The ControlSystem class is designed to initialize with a machine learning model, a specified number of predictive steps (defaulting to 5) and maintains a control history as a list of control actions. It encompasses methods to update the learning rate, η, ensuring it does not fall below 0.001 while also providing functionality to adjust activation functions for dense layers within the model, facilitating systematic modifications in response to dynamic operational conditions.

```
102  class ControlSystem:
103      def __init__(self, model, predictive_steps=5):
104          self.model = model
105          self.predictive_steps = predictive_steps
106          self.control_history = []
107
108      def update_learning_rate(self, delta_lr):
109          current_lr = self.model.optimizer.lr.numpy()
110          new_lr = max(0.001, current_lr + delta_lr)
111          self.model.optimizer.lr.assign(new_lr)
112
113      def update_activation_functions(self, activation_mapping):
114          for layer in self.model.layers:
115              if isinstance(layer, tf.keras.layers.Dense):
116                  current_activation = layer.get_config().get("activation")
117                  new_activation = activation_mapping.get(current_activation)
118                  if new_activation is not None:
119                      layer.activation = tf.keras.activations.get(new_activation)
```

Initialization (init): Takes in:

- **model:** The machine learning model.
- **predictive steps (default = 5):** Number of prediction steps.
- **control history:** Maintains a list of control actions.

Update learning rate: Adjusts the learning rate, η, using:

$$\eta_{new} = \max(0.001, \eta_{current} + \delta_{\eta})$$

Update activation functions: Updates activation functions for dense layers using a mapping.

5.5.2 Predictive Control Function

Performs predictive control utilizing model predictions over specified timesteps.

```python
def predictive_control(self, X):
    """
    Perform predictive control using the model's predictions for the next several timesteps.

    Parameters:
        X (np.ndarray): Input array.

    """
    predictions = self.model.predict(X)

    # Data-Driven Iterative Learning Predictive Control logic
    for step in range(self.predictive_steps):
        prediction = predictions[step]

        # Update learning rate based on historical data
        if self.control_history:
            last_control = self.control_history[-1]
            if prediction > 0.8 and last_control < 0.8:
                self.update_learning_rate(-0.001)
            elif prediction < 0.2 and last_control > 0.2:
                self.update_learning_rate(0.001)

        # Update control history
        self.control_history.append(prediction)

        # Keep history size manageable
        if len(self.control_history) > 1000:
            self.control_history.pop(0)
```

Parameters:

- X: Input array.

Procedure:

(1) Acquire predictions using model.predict(X)

(2) Iterate over the range provided by predictive steps. For each step:

   a. Update learning rate based on historical data if certain conditions are met.

   b. Append prediction to control history.

(3) Limit control history to a size of 1000.

5.5.3 Update Model Function

Updates the learning rate and activation functions of the model.

```python
def update_model(self, u):
    self.update_learning_rate(u[0])

    activation_mapping = {
        'relu': 'sigmoid',
        'sigmoid': 'softmax',
    }
    self.update_activation_functions(activation_mapping)
```

Parameters:

- u: Input list where the first element is used to update the learning rate.

Procedure:

(1) Call ***update_learning_rate*** with the first element of u as argument.

(2) Define an activation mapping:

```
activation_mapping = {
    'relu': 'sigmoid',
    'sigmoid': 'softmax'
}
```

(3) Call ***update_activation_functions*** using the defined mapping.

Before Defense: The simulation initially represents the behavior of two sensors, termed as ”Real” and ”Twin”, over a predefined time. As the simulation progresses, it’s evident that both sensors exhibit minor fluctuations, making them almost identical in their readings. However, during a certain time frame, the real sensor is subjected to a hyperparameter attack. This induces pronounced and erratic fluctuations in the readings of the real sensors for both temperature (T) and motion (M), thus deviating significantly from the corresponding twin sensor readings. The effect of this attack is immediately apparent in the plotted graphs as the real sensor’s waveform diverges from the twin sensors.

After Defense: Post the hyperparameter attack, a defense mechanism is applied to the real sensor readings. This defense, in the form of a dampening strategy, applies a simple moving average on the real sensor data. The aim is to smoothen the abrupt changes brought about by the attack and bring the waveform back in alignment with the twin sensor. As a result of this strategy, the plotted graphs show a convergence of the real and twin sensor readings post-attack. Furthermore, an anomaly detection method is deployed which marks the discrepancies between the real and twin sensor data that surpass a threshold. The defense not only helps in mitigating the effects of the attack but also aids in the identification of potential anomalies during the entire simulation duration.

5.5.4 Before and After Defense Sensor Measurements

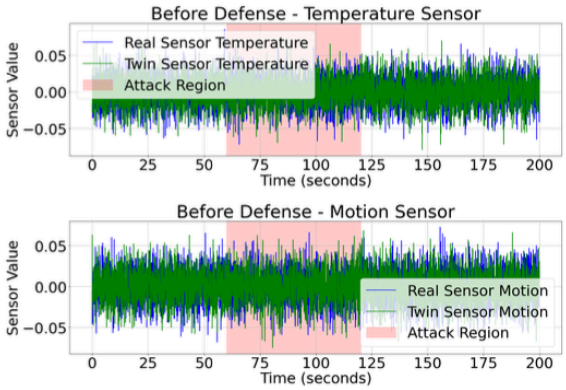


Figure 15: Before Defense: A Digital Twin-Assisted Hyperparameter Attack on Autonomous Control System

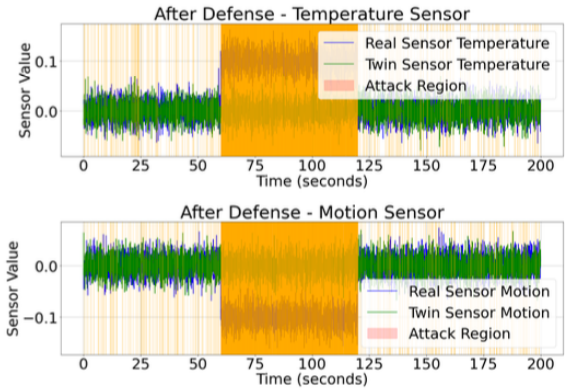


Figure 16: After Defense: Predictive Control for Model with Hyperparameter Adjustment and Defense Mechanism against Hyperparameter Attacks

6. Conclusions and Open Issues

This research delves into hyperparameter attacks on machine learning models tailored for digital twins of nuclear power plants. To counteract the challenges posed by these attacks, an adaptive predictive control strategy anchored on an event-triggering law has been introduced. This predictive control utilizes a sliding window approach, archiving past data and forecasting future outcomes, making it adept at handling the complexities of time-varying multi-rate hyperparameter attacks on machine learning forecasting models. Rigorous simulation studies showcase the effectiveness and robustness of the proposed method, offering valuable insights for enhancing security and resilience in digital twin implementations within critical infrastructures. Open issues include:

Identification and Response to Multiple Hyperparameter Attacks

Addressing the existence of multiple hyperparameter attacks within a machine learning model in the nuclear sector requires stringent methods for identification and tailored response strategies. For Nuclear Regulators, establishing policies that mandate the reporting of identified hyperparameter attacks and the implementation of standardized defense mechanisms becomes imperative. Power Plant Operators must prioritize investing in research and development to devise methods for detecting subtle, specific, or coordinated hyperparameter attacks and implement secure coding practices to mitigate their impacts. Cyber Defense Teams should focus on developing and regularly updating robust defense strategies, ensuring they are apt for addressing the wide array of potential hyperparameter attacks and facilitating continuous monitoring to promptly spot and handle any anomalies.

Enhancing Hyperparameter Attack Models with Decision Matrix Integration

Integrating dynamic decision matrices and time-varying multi-rate functions to address various attack vectors, especially in critical infrastructures like nuclear power plants, is vital. Nuclear Regulators should necessitate the integration of advanced mathematical and computational models, such as decision matrices, into system monitoring to enhance the detection of hyperparameter attacks. Power Plant Operators are recommended to incorporate time-variant multi-rate functions in system modeling, ensuring adaptability to fluctuations and providing a buffer against sudden architectural alterations induced by attacks. Simultaneously, Cyber Defense Teams must continuously work on formulating adaptive algorithms that automatically update the decision matrix in response to evolving attack vectors, thereby safeguarding the models against subtle and drastic adversarial interventions.

Implementing Advanced Control Inputs for Anomaly Mitigation

The strategic deployment of advanced control inputs to proactively identify and counteract anomalies originating from hyperparameter attacks is pivotal in safeguarding nuclear facilities. Nuclear Regulators should establish guidelines for regular audits of control systems, ensuring their capability to detect and respond to anomalies linked to hyperparameter manipulations. Power Plant Operators need to implement advanced control strategies that leverage both robust control theory and machine learning, thereby pre-emptively identifying and neutralizing potential attacks before they inflict system-wide damage. Furthermore, Cyber Defense Teams must commit to exploring and adopting the latest technologies and methodologies in control inputs, ensuring a proactive stance in safeguarding critical infrastructures against emerging threats.

Proactive Threat Mitigation through Predictive Control and Machine Learning

Implementing predictive control strategies, which utilize machine learning to forecast system states, is imperative to pre-emptively mitigate the impacts of hyperparameter attacks in nuclear settings. Nuclear Regulators should enforce regulatory frameworks that mandate the implementation of predictive control and machine learning in cybersecurity strategies, thus ensuring that threats are addressed even before they materialize. Power Plant Operators must adopt predictive control systems that utilize machine learning models to forecast future states, thereby ensuring that potential disruptions are identified and mitigated before they evolve into tangible threats. Moreover, Cyber Defense Teams should constantly enhance predictive algorithms, ensuring they remain abreast of evolving threat parameters and are calibrated to predict and counteract them accurately and promptly.

Page intentionally left blank.