

Machine learning approach to fast thermal equilibrationDiego Rengifo ^{*}*Departamento de Física, Universidad de los Andes, Bogotá 111711, Colombia
and Institut für Physik und Astronomie, Technische Universität Berlin, Hardenbergstraße 36, D-10623 Berlin, Germany*Gabriel Téllez [†]*Departamento de Física, Universidad de los Andes, Bogotá 111711, Colombia*

(Received 10 April 2025; accepted 5 June 2025; published 23 June 2025)

We present a method to design driving protocols that achieve fast thermal equilibration of a system of interest using techniques inspired by machine learning training algorithms. For example, consider a Brownian particle manipulated by optical tweezers. The force on the particle can be controlled and adjusted over time, resulting in a driving protocol that transitions the particle from an initial state to a final state. Once the driving protocol has been completed, the system requires additional time to relax to thermal equilibrium. Designing driving protocols that bypass the relaxation period is of interest so that, at the end of the protocol, the system is either in thermal equilibrium or very close to it. Several studies have addressed this problem through reverse engineering methods, which involve prescribing a specific evolution for the probability density function of the system and then deducing the corresponding form of the driving protocol potential. Here we propose a new method that can be applied to more complex systems where reverse engineering is not feasible. We simulate the evolution of a large ensemble of trajectories while tracking the gradients with respect to a parametrization of the driving protocol. The final probability density function is compared to the target equilibrium one. Using machine learning libraries, the gradients are computed via backpropagation and the protocol is iteratively adjusted until the optimal protocol is achieved. We demonstrate the effectiveness of our approach with several examples.

DOI: [10.1103/q18f-ymr2](https://doi.org/10.1103/q18f-ymr2)**I. INTRODUCTION**

An important branch of physics and engineering is control theory [1], whose objective is to design protocols to control the evolution of a given system and drive it to the desired final state. Often these protocols must also satisfy certain constraints, such as optimizing energy costs. In recent decades, technology has advanced to the point where we can manipulate micro-sized systems [2–4], such as colloidal particles with optical tweezers, micro-oscillators, and microscopic active matter. Due to their small size, significant thermal fluctuations affect these systems. As a result, effectively controlling such systems often requires oversight not only of the average state but also of the fluctuations. This has given rise to a research area where the statistical description of the system (e.g., the density matrix or the probability density function in phase space) becomes the central object of control.

Under time-dependent control, the probability distributions of these systems evolve according to the Fokker-Planck

equation. Once the control protocol is completed, the system remains out of equilibrium and requires time to relax to its new equilibrium state. In many cases, rapid control of the system is desirable to bypass the relaxation period. Such protocols are known as engineered swift equilibration [5] when targeting thermal equilibrium, shortcuts to isothermality [6], or generally as swift state-to-state transformations. For a review, see Ref. [7].

Most methods that achieve this control are based on reverse engineering: One imposes a given evolution for the system's probability density function and then deduces the form of the driving protocol potential. This method is effective for single-particle systems. Theoretically, one can replace the proposed probability density function in the Fokker-Planck equation to obtain the form for controlling the external potential. However, it is not clear how to extend this method to many-particle systems. One can use the controlling potential deduced for a single particle as a starting point for many interacting particles, but this does not guarantee full control of all particles [8]. In this work, we propose a method for numerically designing control protocols. We perform stochastic numerical simulations of the system of interest and employ machine learning techniques—in particular, automatic differentiation [9]—to evaluate how changes in control potential affect the statistical state of the system and drive it to the desired final state. This method has some similarities to the image generative diffusion models [10,11] whose objective is to recreate a complex probability distribution from a simple one (normal distribution),

^{*}Contact author: d.rengifo@uniandes.edu.co[†]Contact author: gtellez@uniandes.edu.co

Published by the American Physical Society under the terms of the Creative Commons Attribution 4.0 International license. Further distribution of this work must maintain attribution to the author(s) and the published article's title, journal citation, and DOI.

although the exact implementation has some key differences. Similar techniques have been used to design protocols that optimize the work done on the system [12]. Here, however, we are interested in controlling the final state of the system. A similar goal in the realm of active matter has been explored in Ref. [13] using evolutionary algorithms, which offer a complementary approach to the present work.

The remainder of this article is organized as follows. In Sec. II, we explain in detail our numerical technique. Then, in Sec. III we apply it to the control of a Brownian particle under a variable harmonic potential. This serves as a test of our method as its performance can be compared to analytical methods. In Sec. IV, we apply our method to achieve swift state-to-state transformations of a Brownian particle when the control potential is not harmonic, as an illustration of a situation where analytical methods are more difficult to implement.

II. NUMERICAL DESIGN OF CONTROL PROTOCOLS

To explain our method clearly, we consider a simple system consisting of a Brownian particle with position X that experiences thermal fluctuations at temperature T and is driven by a controlling force $F(X, \lambda(t))$, which depends on the position of the particle and a set of external, time-dependent parameters $\lambda(t)$. The particle is in a viscous fluid that exerts a drag force, characterized by a friction coefficient γ . We assume that the inertial timescale is much shorter than both the relaxation time and the duration of the controlling protocol; therefore, we operate in the overdamped regime. We describe the stochastic dynamics of the particle using a Langevin equation

$$\frac{dX}{dt} = F(X, \lambda(t))/\gamma + \sqrt{2D}\xi(t), \quad (1)$$

where $D = k_B T/\gamma$ is the diffusion coefficient, k_B denotes the Boltzmann constant, and we model $\xi(t)$ as a Gaussian white noise satisfying $\langle \xi(t) \rangle = 0$ and $\langle \xi(t)\xi(t') \rangle = \delta(t - t')$. In the rest of this paper, we work with dimensionless quantities defined as follows. We first identify a timescale τ , which can represent the relaxation time of the system (for a fixed λ), the duration of the controlling protocol, or the total simulation time. Then the time is expressed in dimensionless units as $\hat{t} = t/\tau$, the position as $\hat{x} = x/\sqrt{D\tau}$, the Gaussian noise as $\hat{\xi}(\hat{t}) = \sqrt{\tau}\xi(t)$, and the force as $\hat{F} = F\sqrt{D\tau}/(k_B T)$. The force derives from a potential energy: $F = -\partial_x U(x, \lambda)$, and the potential energy is expressed in dimensionless units as $\hat{U} = U(x, \lambda)/(k_B T)$. With these definitions, the Langevin equation reads

$$\frac{d\hat{X}}{d\hat{t}} = \hat{F}(\hat{X}, \hat{\lambda}(\hat{t})) + \sqrt{2}\hat{\xi}(\hat{t}). \quad (2)$$

For simplicity, we drop the hat in what follows. An equivalent description of the system is through the Fokker-Planck equation for the position probability density function $P(x, t)$, which reads

$$\partial_t P(x, t) = -\partial_x [F(x, \lambda(t))P(x, t)] + \partial_x^2 P(x, t). \quad (3)$$

We now state the problem we want to solve. Assume that the system starts at $t = 0$ in equilibrium with probability

density $P(x, 0) = P_i^{\text{eq}}(x) = \frac{1}{Z_i} e^{-U(x, \lambda_i)}$ for an initial control parameter λ_i . Here $Z_i = \int e^{-U(x, \lambda_i)} dx$ is the corresponding initial partition function. A control protocol $\lambda(t)$ of duration t_f is applied to the system and at $t = t_f$ we wish the system to be in a new thermal equilibrium state corresponding to a final value λ_f of the controlling parameters: $P_f^{\text{eq}}(x) = \frac{1}{Z_f} e^{-U(x, \lambda_f)}$ with $Z_f = \int e^{-U(x, \lambda_f)} dx$. For an arbitrary control protocol $\lambda(t)$, even when we set $\lambda(0^-) = \lambda_i$ and $\lambda(t_f^+) = \lambda_f$, the system lags behind the instantaneous Boltzmann distribution $\propto e^{-U(x, \lambda(t))}$. Consequently, at $t = t_f$, the system is not in thermal equilibrium and requires additional time to relax to the new equilibrium state. Our goal is to design the protocol $\lambda(t)$ so that $P(x, t_f)$ approximates the desired equilibrium $P_f^{\text{eq}}(x) = \frac{1}{Z_f} e^{-U(x, \lambda_f)}$ as closely as possible. We base our method on numerical simulations of the system and on systematic adjustments to the control protocol.

A. Numerical training algorithm

To achieve this, we begin with an arbitrary protocol $\lambda(t)$. For example, we choose a linear interpolation between λ_i and λ_f , possibly adding some random values. We perform $N_s \sim 100\,000$ parallel simulations of the Langevin equation, Eq. (2), on a graphics processing unit (GPU) using the initial protocol $\lambda(t)$. These simulations produce a large ensemble of trajectories $\{x_n(t)\}_{n \in \{1, \dots, N_s\}}$ from which we compute the final probability density function $P(x, t_f)$. We then compare the calculated $P(x, t_f)$ with the desired equilibrium distribution $P_f^{\text{eq}}(x)$. Initially, these two distributions differ significantly. We quantify the difference between $P(x, t_f)$ and $P_f^{\text{eq}}(x)$ using a *loss* function, which we will define later. Thus, we will need adjust the protocol $\lambda(t)$ to minimize the loss function. However, it will be very inefficient to blindly do arbitrary changes to $\lambda(t)$ and rerun the simulations to find how the final distribution has changed. To do this effectively, we will use techniques from machine learning, in particular automatic differentiation [9], which allows us to know in advance in which directions the changes on the protocol will make smaller the loss function between the final distribution and the desired one.

Although automatic differentiation is a well-known technique in the realm of machine learning, we briefly review its key concepts to ensure completeness and clarify the technical challenges we overcame. In a single pass of the system simulation, the final probability density function $P(x, t_f)$ and the loss function result from numerous elementary operations, such as sums and multiplications, performed throughout. Although we cannot analytically compute $P(x, t_f)$, the loss function, or its gradient with respect to λ , we know the partial derivatives of each elementary operation with respect to its inputs. Throughout the simulation, these operations form a computational tree. We store the partial derivatives of each elementary operation along with each intermediate result. Then the computational tree is traced backward and the chain rule is applied to compute the gradient of the loss function with respect to the control protocol λ . This technique is known as backpropagation in automatic differentiation.

For example, a single step of the resolution of the Langevin equation, using the Euler-Maruyama method, in the Itô dis-

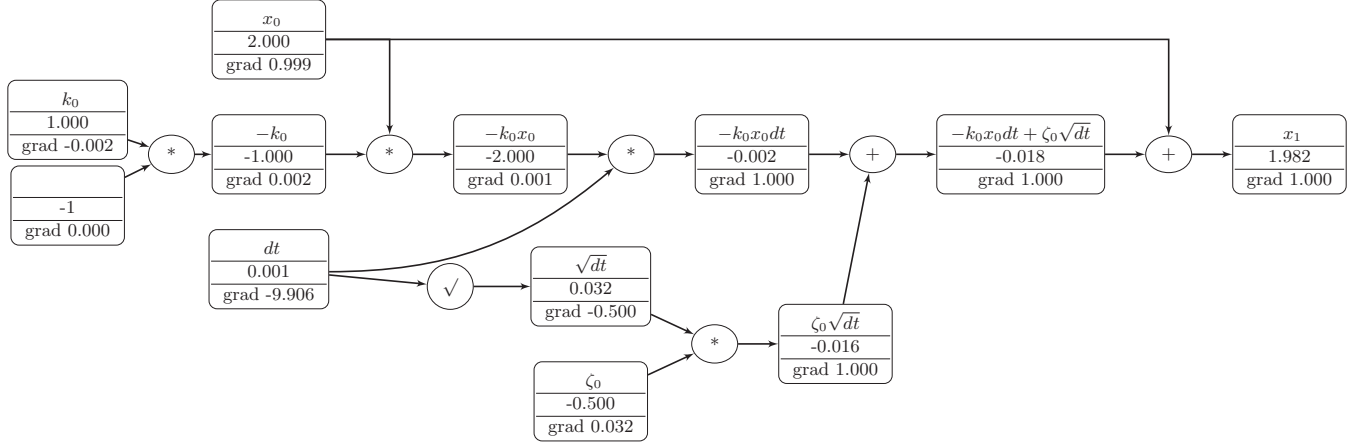


FIG. 1. Computational graph for a single step of the Langevin equation, Eq. (4), and its corresponding gradients. In this example, we used numerical values $x_0 = 2$, $k_0 = 1$, $dt = 0.001$, and $\zeta_0 = -0.5$ (a particular realization drawn from a normal distribution $\mathcal{N}(0, \sqrt{2})$). The graph is composed of boxes (rectangles) and operations (ellipses). Each box has three subdivisions indicating the variable, its value, and the gradient of x_1 with respect to the variable in the actual box, respectively.

cretization, can be written as

$$x_1 = x_0 - k_0 x_0 dt + \zeta_0 \sqrt{dt} \quad (4)$$

for a force $F = -k_0 x$ and where $\zeta_0 \sim \mathcal{N}(0, \sqrt{2})$. The particle moves from an initial position x_0 to a position x_1 in a time dt . Figure 1 shows the computational tree for this operation. From the initial set of parameters (x_0 , k_0 , etc.), the values are calculated forward, and the gradients are calculated backward. Suppose that the stiffness k_0 is one of the adjustable parameters. In Fig. 1, we choose $k_0 = 1$, $x_0 = 2$, and $dt = 0.001$ as inputs. Multiplications, sums, and square-root operations are performed as shown in the computational graph to obtain the value of the output $x_1 = 1.982$. To compute the gradient of the output of Eq. (4), x_1 , with respect to k_0 , $\frac{\partial x_1}{\partial k_0}$, we analyze how changes in k_0 propagate through the computational tree in the backward direction. We start in the last box of the computational tree in Fig. 1 where the value of x_1 is reported and the gradient $\partial x_1 / \partial x_1 = 1$ is stored. The previous operation is a sum. The partial derivative of the output of the sum with respect to each of its inputs is 1, so the gradient of x_1 with respect to $-k_0 x_0 dt + \zeta_0 \sqrt{dt}$ is 1 as shown in the preceding box.

Continuing the backward pass, we encounter another sum, again with a gradient of 1. We arrive at a multiplication between dt and $-k_0 x_0$. The partial derivative of the result of this multiplication with respect to $-k_0 x_0$ is simply dt , the other factor in the multiplication. In this example, we choose $dt = 0.001$, and therefore the gradient of x_1 with respect to $-k_0 x_0$ is now 0.001 as shown in the box labeled $-k_0 x_0$. Continuing backwards, we reach a multiplication between $-k_0$ by x_0 . The partial derivative $-k_0 x_0$ with respect to $-k_0$ is $x_0 = 2$. Applying the chain rule, we multiply this partial derivative, 2, with the accumulated gradient, 0.001, to obtain the gradient of x_1 with respect to $-k_0$: $2 \times 0.001 = 0.002$. The last operation is a multiplication of -1 by k_0 , which finally gives the desired gradient $\partial x_1 / \partial k_0 = -0.002$. The backpropagation process efficiently tracks these contributions by multiplying local derivatives along the computational path, demonstrating how automatic differentiation captures the sensitivity of x_1 to

small changes in k_0 . This method is significantly more efficient than naive finite differences, which require recomputing x_1 for each perturbation of k_0 .

In this previous example, k_0 appears only in one branch of the computational tree which simplifies the analysis of the backward propagation of the gradient. When a parameter appears on more branches, one needs to add each gradient contribution when they merge. For example, in the x_0 box, we obtain $\partial x_1 / \partial x_0 = 0.999$ by adding the gradient contributions from two branches: 1 and -0.001 .

For a review of automatic differentiation, see Ref. [9]; an excellent tutorial appears in Ref. [14]. In practice, we did not implement automatic differentiation ourselves; instead, we used an existing machine learning library optimized for parallel GPU computing. We used pytorch [15]; however, our algorithm could also be easily ported to other frameworks such as tensorflow [16] or jax [17].

After making the necessary changes in the control protocol $\lambda(t)$ in the direction provided by the gradients (gradient descent method), the algorithm starts over to perform a new set of simulations, evaluates again the loss function, and adjusts the protocol until the loss function converges to the smallest possible value. Each pass of simulations, computing the gradients and adjusting the protocol is called an epoch following the machine learning terminology.

This algorithm is very similar to the one on which neural network training is based; therefore, the use of a dedicated library such as pytorch is very convenient. In a sense, it is a particular form of a neural network composed of $N_T = t_f / dt$ layers, each of which performs a single step in the simulation:

$$x(t + dt) = x(t) + F(x(t), \lambda(t))dt + \sqrt{2}dW(t) \quad (5)$$

with $W(t)$ a Wiener process. The network parameters are encoded in $\lambda(t)$. This neural network takes as input the initial samples $\{x_n(0)\}_{n \in \{1, \dots, N_s\}}$ sampled from a Boltzmann distribution $e^{-U(x, \lambda_i)} / Z_i$. In parallel, each sample goes through the layers performing the calculations given by Eq. (5) to obtain after N_T layers the final samples $\{x_n(t_f)\}_{n \in \{1, \dots, N_s\}}$. Note that, at this stage, each sample is independent of the rest. Then a final

layer takes all the samples and combines them to compute the loss function L (see Sec. II B for details). The gradients $\partial_{\lambda(t)}L$ of the loss function with respect to the parameters $\lambda(t)$ are computed using backpropagation, and the parameters are updated accordingly:

$$\lambda(t) \leftarrow \lambda(t) - \eta \partial_{\lambda(t)}L, \quad (6)$$

where η is the learning rate. The precise meaning of $\partial_{\lambda(t)}L$ and Eq. (6) depends on the way the protocol is parametrized; see Sec. II C for details. This process is repeated until the loss function converges to a minimum.

B. Choice of the loss function

Let us now discuss in more detail some important technical aspects of the algorithm, starting with the choice of the loss function. A first natural choice would be to use the Kullback-Leibler divergence,

$$D(P(x, t_f) || P_f^{\text{eq}}) = \int P(x, t_f) \ln \left[\frac{P(x, t_f)}{P_f^{\text{eq}}(x)} \right] dx. \quad (7)$$

This will require computing $P(x, t_f)$ from the set of samples $\{x(t_f)\}$ obtained from the simulation at the final time t_f . Numerically, this could be done by building a histogram. Unfortunately, creating histograms requires selecting bin sizes, a step that is not differentiable and lacks support for automatic differentiation. For example, in `pytorch`, the `torch.histogram` function does not have a backward method implemented to compute the gradients. Therefore, a different choice of loss function is needed.

For the case where the force exerted on the particle derives from a harmonic potential (Sec. II), one can show that if the initial probability distribution is Gaussian, it will remain Gaussian at all times. In this case a simple loss function can be chosen by comparing the average value of the position

$$L_{\text{mean}} = (\langle x(t_f) \rangle - \langle x \rangle_f^{\text{eq}})^2, \quad (8)$$

and/or its variance

$$L_{\text{var}} = (\langle \Delta x(t_f)^2 \rangle - \langle \Delta x^2 \rangle_f^{\text{eq}})^2. \quad (9)$$

Here, the average

$$\langle x(t_f) \rangle = \int x P(x, t_f) dx \quad (10)$$

$$\simeq \frac{1}{N_s} \sum_{n=1}^{N_s} x_n(t_f) \quad (11)$$

is computed from the set of samples of the simulation, and the equilibrium one,

$$\langle x \rangle_f^{\text{eq}} = \int x P_f^{\text{eq}}(x) dx, \quad (12)$$

is known directly from the analytic expression of $P_f^{\text{eq}}(x)$. For the variance, we have defined $\Delta x(t_f) = x(t_f) - \langle x(t_f) \rangle$ and $\Delta x = x - \langle x \rangle_f^{\text{eq}}$. These loss functions defined in Eqs. (8) and (9) are differentiable and can be used to adjust the control protocol.

However, beyond the harmonic case, it is not enough to verify that the final average and variance converge to the

expected values. One could compare higher-order cumulants or, better yet, compare the characteristic functions of the final distribution

$$\tilde{P}(k, t_f) = \langle e^{ikx(t_f)} \rangle \quad (13)$$

with the expected one

$$\tilde{P}_f^{\text{eq}}(k) = \int P_f^{\text{eq}}(x) e^{ikx} dx. \quad (14)$$

Contrary to the nondifferentiable computation of $P(x, t_f)$ by building a histogram, the computation of the characteristic function is a differentiable operation from the samples $\{x_n(t_f)\}_{n \in \{1, \dots, N_s\}}$ as it involves computing the exponential function and summing over

$$\tilde{P}(k, t_f) \simeq \frac{1}{N_s} \sum_{n=1}^{N_s} e^{ikx_n(t_f)}. \quad (15)$$

For this more general case, the loss function is

$$L_{\text{char}} = \int |\tilde{P}(k, t_f) - \tilde{P}_f^{\text{eq}}(k)|^2 dk. \quad (16)$$

Numerically, this is approximated by choosing an appropriate range of values for k and replacing the integral by a sum over those values of k .

C. Parametrization of the protocol

The control protocol $\lambda(t)$ can be parametrized in many ways. For example, $\lambda(t)$ can be a piecewise linear function, a polynomial, a rational function, etc. The choice of the parametrization will depend on the specific problem at hand. For example, if the control protocol is expected to be smooth, a polynomial or a rational function parametrization could be a good choice. In our examples, we explore a few possibilities: a piecewise linear parameterization, a polynomial parameterization, and a neural network to approximate $\lambda(t)$.

The piecewise linear parametrization is the most intuitive one. The duration of the protocol $[0, t_f]$ is divided into N_p time intervals of equal duration: $0 = t_0 < t_1 < \dots < t_{N_p} = t_f$, with $t_j = j \Delta t$ where $\Delta t = t_f / N_p$. The value of $\lambda(t_j) = \lambda_j$ at the beginning of each interval is a parameter of the protocol. Between times t_j and t_{j+1} , $\lambda(t)$ is a linear interpolation between λ_j and λ_{j+1} . The total number of parameters is N_p . The protocol is then defined by the set of parameters $\{\lambda_1, \dots, \lambda_{N_p}\}$ that are the variables to be adjusted to minimize the loss function. After each simulation, the gradients of the loss function L with respect to the parameters λ_j are calculated and the new values of the parameters are updated accordingly to $\lambda_j \leftarrow \lambda_j - \eta \partial_{\lambda_j} L$ where η is the learning rate.

In a polynomial parametrization, $\lambda(t)$ is a polynomial function of time:

$$\lambda(t) = \sum_{n=0}^P a_n t^n. \quad (17)$$

The coefficients a_n are the variables that the algorithm will optimize. After each simulation, these are updated using gradient descent: $a_n \leftarrow a_n - \eta \partial_{a_n} L$.

Finally, based on the universal approximation theorem [18], we also explored a neural network parametrization

of $\lambda(t)$. We selected a feedforward neural network architecture, which can be implemented using PyTorch. The input layer consists of a single neuron representing time, t . This neuron processes t through an activation function and passes the result to the next layer, which contains n neurons. Each neuron in this layer applies an activation function to the input it receives and forwards the output to subsequent layers. This process continues until the final layer whose output is of dimension one determining the output of the model $\lambda(t)$. An example of this neural network parametrization of the protocol is shown in Sec. III B 1.

D. Relation with previous works

In recent years, there has been a growing interest in applying machine learning techniques in statistical physics, from the control and design of self-assembling materials [19,20], training feedback protocols (Maxwell demons) to extract work and absorb entropy [21], doing memory erasure [22,23], and optimizing the control of nonequilibrium systems [12,13,24]. This interplay has been fruitful both ways, as concepts from statistical physics have open the way for new techniques in machine learning, for example in diffusion generative models [10,11,25,26] or training neural networks using Monte Carlo methods [27].

Our method is inspired by previous work [12] on optimal control of nonequilibrium systems. In that work, stochastic simulations of a system of interest are performed in a differentiable way to compute the gradients of a loss function with respect to the control protocol. The loss function used in Ref. [12] was the entropy production or the work done on the system. Our goal here is different; the loss function we use is to minimize the difference between the final probability density function of the system and the desired equilibrium distribution. Our algorithm implementation is independent from Ref. [12] using pytorch instead of jax.

The optimal control of stochastic systems and driving them from an initial state to a target final state can also be solved by deterministic optimal transport techniques as shown in Ref. [28], where the stochastic thermodynamics optimization of heat or work problem is mapped into a Burgers equation. Then standard optimal transport techniques [29] can be used to find optimal control protocols [30–32]. Our present work is an alternative to those methods.

In the realm of machine learning, there has been recent interest in neural ordinary differential equations [33], where the dynamics $\{x(t)\}$ of a system is given by an ordinary differential equation

$$\frac{dx}{dt} = f(x(t), t, \lambda), \quad (18)$$

and f is described by a neural network, with parameters λ that are to be optimized such that the system is in a desired final state $x(t_f)$. In Ref. [33], a method was devised to compute the gradients of the loss function without backpropagating through all the intermediate N_T steps. The method is known as the adjoint sensitivity method [34] and is quite memory efficient of $\mathcal{O}(1)$ instead of $\mathcal{O}(N_T)$ for the backpropagation method. Unfortunately, it is not clear how to apply this method to stochastic differential equations such as

the one we are solving. However, a promising proposal to extend the adjoint sensitivity method to deal with gradient computations for stochastic differential equations in a memory efficient manner of order $\mathcal{O}(1)$ was proposed in Ref. [35], at the expense of a longer time computation $\mathcal{O}(N_T \log N_T)$. However, we did not need to implement this method, as the backpropagation method through all intermediate computations was enough for our purposes. Since our objective is to find a control protocol to achieve fast thermalization in a short time duration, the number of steps N_T is not very large, and the memory bottleneck was not reached in our simulations.

III. BROWNIAN PARTICLE UNDER HARMONIC POTENTIALS

In this section, we examine an overdamped Brownian particle subject to a harmonic potential, the general expression for which is

$$U(x, \lambda(t)) = \frac{1}{2}k(t)(x - c(t))^2. \quad (19)$$

The external control parameter $\lambda(t)$ can represent here the potential center $c(t)$, the stiffness $k(t)$, or both. The core concept of optimal control involves determining a form of λ that allows the system to optimize or reduce a specific desired property. In the following, the focus is on control protocols that speed up the equilibration time or optimize the work performed on the particle or aim at both goals simultaneously.

A. Changing the stiffness of the harmonic potential

First, we analyze a system with a fixed center, $c(t) = 0$, and a time-dependent stiffness, $k(t)$, subject to the boundary conditions $k(0^-) = k_i$ and $k(t_f^+) = k_f$. We express the system in dimensionless units based on the natural relaxation time $\tau = \gamma/k_f$, which depends on the final stiffness k_f , as outlined in Sec. I. This is the relaxation time if an abrupt quench, where the potential transforms instantaneously from the initial to the final potential is performed. We seek a control protocol that satisfies the boundary conditions while ensuring that $t_f < \tau$.

Researchers have proposed several analytical protocols to accelerate thermal equilibration in this system [5,7,36]. These protocols provide a theoretical framework for optimizing control strategies that reduce relaxation times beyond what passive equilibration allows. Thus, replicating these well-known results provides a valuable starting point to validate our approach before applying it to more complex problems.

1. One parameter protocol: The two-step protocol

The simplest and most intuitive protocol is likely the two-step protocol (TSP) [36], where the stiffness changes abruptly from k_i to some value k_m at $t = 0$ and then from k_m to k_f at $t = t_f$. The value of k_m is increased sufficiently to ensure that the particle relaxes faster to the new equilibrium state at t_f .

Using the Fokker-Planck equation (3) with the potential Eq. (19), one can obtain the evolution equation for all cumulants χ_m of the probability density function of the position. From there, one can show that if the system starts with an initial probability density function that is Gaussian and the driving protocol is harmonic, all cumulants except the second

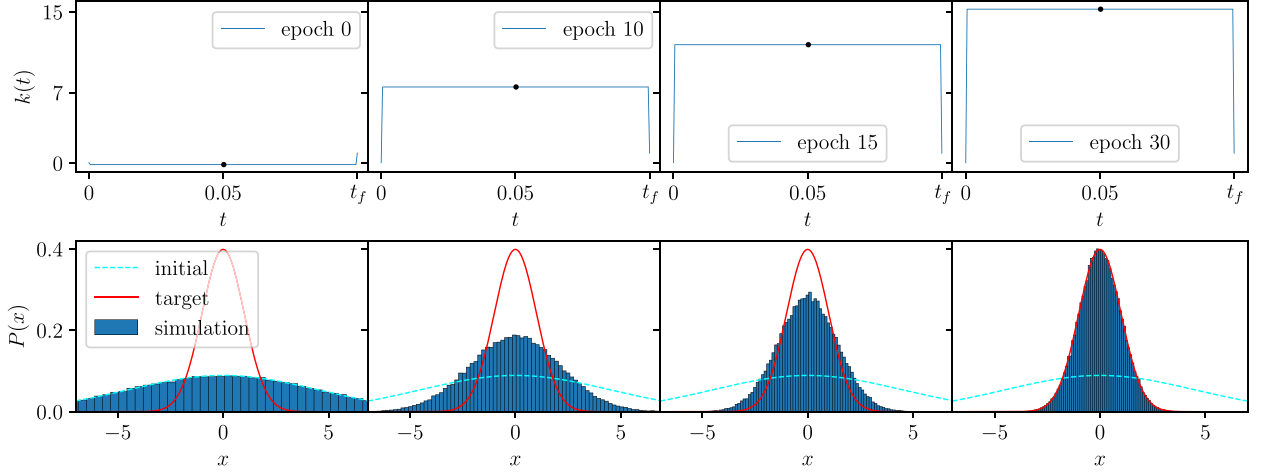


FIG. 2. Training progress over 30 epochs for the two-step protocol applied to the stiffness parameter, utilizing the loss function defined in Eq. (16) and a learning rate of $\eta = 100$. The initial (dashed) and target (solid) probability distributions are displayed, alongside a histogram generated from the model trained up to the specified epoch. The system parameters are set to $k_i = 0.05$, $k_f = 1.0$, and $t_f = 0.1$.

one (the variance) will be null at all times. This means that the probability density function will remain Gaussian at all times, with a time-dependent variance $\chi_2(t)$ satisfying

$$\frac{d\chi_2(t)}{dt} + 2k(t)\chi_2(t) = 2. \quad (20)$$

To achieve thermalization, the variance at the final time t_f must be $\chi_2(t_f) = 1/k_f$ so that $P(x, t_f) = P_f^{\text{eq}}(x) = e^{-k_f x^2/2} / \sqrt{2\pi/k_f}$. Since the stiffness is constant in the time interval $(0, t_f)$, $x(t)$ is an Ornstein-Uhlenbeck process whose probability density function $P(x, t)$ is a Gaussian with zero mean and variance

$$\chi_2(t) = \frac{1}{k_m} + \left(\frac{1}{k_i} - \frac{1}{k_m} \right) e^{-2k_m t}. \quad (21)$$

Using $\chi_2(t_f) = 1/k_f$, we obtain the following transcendental equation for k_m (for details see Ref. [36]):

$$\frac{1}{k_f} = \frac{1}{k_m} + \left(\frac{1}{k_i} - \frac{1}{k_m} \right) e^{-2k_m t_f}. \quad (22)$$

This analytic solution of the problem can be used as a test of our numerical training algorithm. We use a parametrization of $\kappa(t)$ with a constant value k_m between $(0, t_f)$ and discontinuous at $t = 0$ and $t = t_f$. This protocol has only one adjustable parameter which is k_m . The loss function can be either the variance loss function, Eq. (9), or the characteristic function loss function, Eq. (16).

For numerical calculations, we used the values $k_i = 0.05$ and $k_f = 1.0$, for which the relaxation time $\tau = 1/k_f = 1.0$. We aim to achieve equilibration at time $t_f = 0.1$, an order of magnitude faster than the natural relaxation. We trained this one-parameter model during 30 epochs with a learning rate of $\eta = 100.0$. This large value (compared to the traditional values used in machine learning of order 10^{-2}) is understood because there is only one parameter to adjust. For training, we used the characteristic function loss function in Eq. (16). In Fig. 2, we show the training results for the stiffness as a function of time after some epochs. This plot also includes

the probability density function (PDF) of the position, at the initial time, the target PDF, and the position histogram after simulating the system with the protocol given by the respective epoch. During the first 10 epochs, the system lags behind the final equilibrium distribution at t_f . But after epoch 10, the adjustment made to k_m drives the system faster and closer to the expected final equilibrium distribution. By epoch 30, the final probability density is indistinguishable from the expected one, where the loss function was of order 10^{-5} . The value obtained from k_m with this training was $k_m = 15.27$, while the theoretical value from Eq. (22) is $k_m = 15.30$. This corresponds to a precision of 0.20% relative to the expected value.

2. Linear interpolation protocols

We now consider a more general case where the protocol $k(t)$ is not constant in the interval $(0, t_f)$ and is continuous at the boundary. In this case, we can parametrize k using piecewise linear parametrization with $N_p = 10$ parameters. It is important to recall that the time interval $[0, t_f]$ is divided into N_p subintervals and the adjustable parameters are the values of $k(t_j)$ at the beginning of each subinterval as explained in Sec. II C. For the training process, we took the initial and final stiffness to be $k_i = 0.5$ and $k_f = 1$, respectively. These values lead to a relaxation time $\tau = 1$ and we set the duration of the protocol to be $t_f = 1/30$. Figure 3 shows results for two trained protocols and the TSP for reference. The protocol represented by the red curve was obtained through optimization with the characteristic function loss Eq. (16) during 60 epochs. Notice that this protocol has a very fast variation at the initial $t_i = 0$ and final times t_f and does not change much during the rest of the time interval. It quite resembles the corresponding two-step protocol shown in green. There are an infinite number of solutions to the thermalization problem; nevertheless, this shows that in the training algorithm the two-step protocol plays an important role as it exhibits a strong and stable attraction for the algorithm.

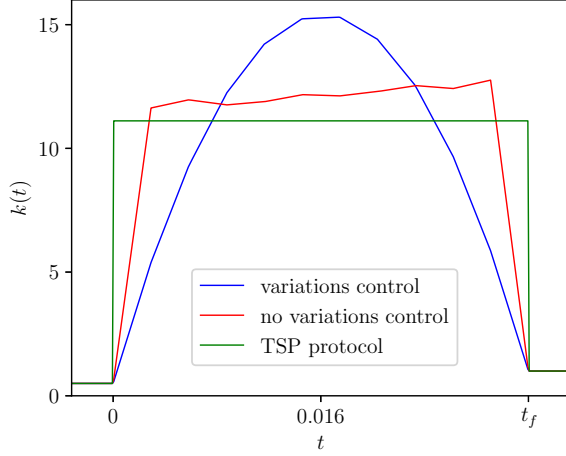


FIG. 3. Stiffness as a function of time designed to accelerate equilibration. The red curve represents a 10-parameter protocol trained using the characteristic function loss, Eq. (16). The blue curve also corresponds to a 10-parameter protocol but includes an additional penalty term in the loss function to suppress large variations in stiffness, as given in Eq. (33). For reference, the green curve depicts the corresponding two-step protocol. The initial and final stiffness values are set to $k_i = 0.5$ and $k_f = 1$, respectively, with a protocol duration of $t_f = 1/30$.

In a practical implementation of the protocol, it might not be desirable to have very large variations in the stiffness in a short time or discontinuities in it. To achieve a smoother protocol, we controlled the variations of $k(t)$ between successive time intervals by adding to the loss function a term penalizing strong variations:

$$L_{\text{gradk}} = \sum_{j=0}^{N_p-1} (k(t_{j+1}) - k(t_j))^2. \quad (23)$$

Then, as a loss function, we can adopt the combination $L = (1 - \alpha)L_{\text{char}} + \alpha L_{\text{gradk}}$, where α is a mixture parameter which we initially fixed at $\alpha = 10^{-4}$ for 60 epochs, then turned it off at $\alpha = 0$ for another 60 epochs of training. This procedure results in much smoother stiffness as shown in blue in Fig. 3. Introducing restrictions or desirable properties into the loss function is a crucial tool, as it enables the customization of control protocols to exhibit convenient and tailored properties.

It can be observed that all three protocols in Fig. 3 require a large increase in the mean value of $k(t)$ to shorten the effective relaxation time of the protocol. This is a common feature of all protocols that has been analyzed in Ref. [36]. The average value of $k(t)$ during the duration of the protocol is of the order $\frac{1}{2t_f} \ln \frac{k_f}{k_i}$ for $t_f \ll 1$.

3. Polynomial parametrization

The piecewise linear parametrization model might initially seem restrictive due to its reliance on linear interpolation. However, during the optimization of the loss function, nonlinearities are inherently accounted for. In addition, increasing the number of parameters can improve the model's ability to reproduce more complex protocols. Alternatively, models can be defined to incorporate nonlinearities from the outset.

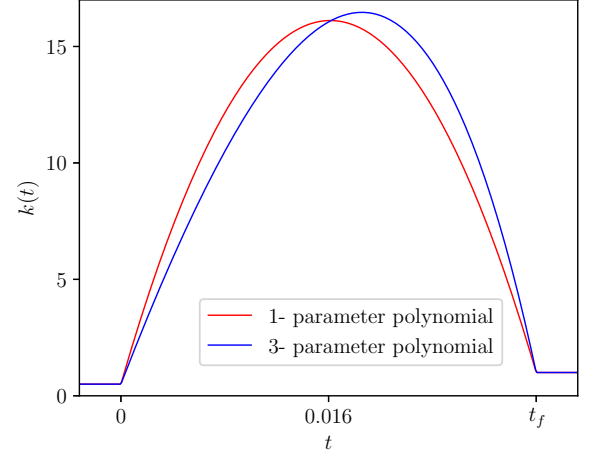


FIG. 4. Results for stiffness $k(t)$ after training two models using the polynomial parametrization defined in Eq. (24). The red curve represents the model trained with a single parameter, while the blue curve corresponds to the model trained with three parameters. The initial and final stiffness values are $k_i = 0.5$ and $k_f = 1$, respectively, with a protocol duration of $t_f = 1/30$.

For instance, in our system, the stiffness can be parameterized using a polynomial, with its coefficients serving as trainable parameters.

Consider a polynomial parametrization for the stiffness given by

$$k(t) = k_i + s(k_f - k_i) + s(1 - s)p(s), \quad (24)$$

where $s = t/t_f$ and $p(s)$ is a polynomial

$$p(s) = \sum_{n=0}^{N_p-1} a_n s^n. \quad (25)$$

The coefficients a_n are the parameters that the algorithm will optimize. The form proposed for the stiffness, Eq. (24), ensures continuity at $t = 0$ and $t = t_f$. Figure 4 shows two trained protocols, one with a single coefficient (one parameter, p has degree 0) and the other with three coefficients (three parameters, p has degree 2). The initial and final stiffnesses are $k_i = 0.5$ and $k_f = 1$, and the duration of the protocol is $t_f = 1/30$. Both protocols were trained for 60 epochs and the loss function used was the characteristic loss function Eq. (16), which at the end of training was of the order 10^{-5} . For the one-parameter protocol, the parameter value was $a_0 = 61.69$, and for the three-parameter protocol: $a_0 = 45.84$, $a_1 = 23.51$, and $a_2 = 14.54$.

Although the number of parameters N_p can be chosen arbitrarily, for this particular problem, with a harmonic potential, one can restrict the search of a solution to a one-dimensional space. This is so because the probability density function remains Gaussian at all times, with a time-dependent variance $\chi_2(t)$ that satisfies Eq. (20). To achieve a fast thermalization, the variance at the final time t_f needs to be $1/k_f$. Therefore, one can parametrize $\kappa(t)$ with only one parameter that get fixed by the condition $\chi_2(t_f) = 1/k_f$. However, for more complex potentials, such as the quartic potential discussed in Sec. III, the search space must be expanded, necessitating an increase in the number of parameters. This is because addi-

tional cumulants, χ_m , become nonzero, and their evolution is governed by a hierarchy of coupled differential equations that must be solved simultaneously.

B. Fast thermal protocol that optimize work

1. Optimizing work alone

In the previous examples, we focused on demonstrating how to train an algorithm capable of achieving fast thermalization. Now, we aim to show that the same algorithm also performs exceptionally well in identifying protocols that optimize the work done on the system. In the realm of stochastic thermodynamics [37], the work done on a system is a random variable described by

$$W = \int_0^{t_f} \dot{\lambda}(t) \frac{\partial U}{\partial \lambda}(x, \lambda(t)) dt, \quad (26)$$

where $\dot{\lambda}(t)$ represents the time derivative of the control parameter $\lambda(t)$. The average work done on the system can be used as a loss function

$$L_W = \langle W \rangle = \int_0^{t_f} dt \dot{\lambda}(t) \left\langle \frac{\partial U}{\partial \lambda}(x, \lambda(t)) \right\rangle = \frac{1}{2} \int_0^{t_f} dt \dot{k}(t) \langle x^2 \rangle, \quad (27)$$

where the average $\langle \cdot \rangle$ is taken over the initial distribution and noise realizations. The last equality is valid only for the harmonic oscillator with $\lambda(t) = k(t)$. The problem of determining the optimal work protocol was addressed in Ref. [38], where mathematical expressions were derived for the optimal stiffness and center protocols. For reference, we include their results here. The optimal stiffness protocol within the interval $0 < t < t_f$ is given by:

$$k_{\text{opt}}(t) = \frac{k_i - A(1 + At)}{(1 + At)^2}, \quad (28)$$

and the optimal work is

$$\langle W \rangle_{\text{opt}} = \frac{(At_f)^2}{k_i t_f} - \ln(1 + At_f) + \frac{1}{2} \frac{k_f}{k_i} (1 + At_f)^2 - \frac{1}{2}, \quad (29)$$

where the constant A is

$$A = \frac{-1 - k_f t_f + \sqrt{1 + 2k_i t_f + k_i k_f t_f^2}}{t_f(2 + k_f t_f)}. \quad (30)$$

To reproduce this result, a piecewise linear parametrization model with $N_p = 10$ learnable parameters was trained over 60 epochs and a learning rate of $\eta = 10.0$. For the numerical simulation, the parameters were $k_i = 1.0$, $k_f = 5.0$, and $t_f = 1.0$. From Eq. (29), the expected work is $\langle W \rangle_{\text{opt}} = 1.0565$. The value obtained after 60 epochs of training is $\langle W \rangle = 1.0594$, corresponding to a relative error of 0.27%. This shows that the model accurately reproduces the optimal protocol for stiffness using average work as the loss function. The trained protocol and the analytical solution Eq. (28) as well as the loss function are shown in Fig. 5.

We can also work with parametrizations that involve neural networks. The protocol described in Eq. (28) can be reproduced by a neural network whose architecture is as follows. The input t is first normalized by dividing it by the time duration parameter t_f . The normalized input is then passed

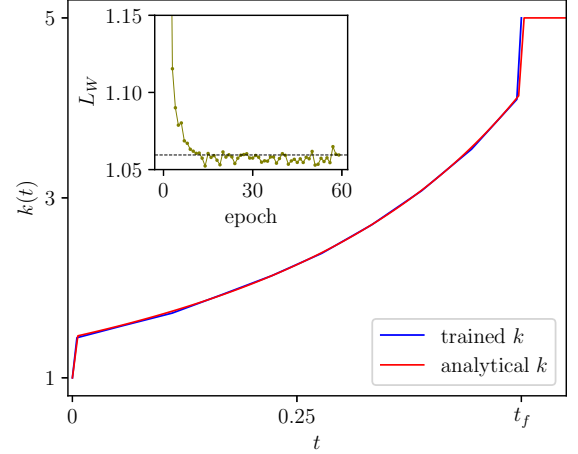


FIG. 5. Stiffness as a function of time for the optimal work protocol (red curve), given by Eq. (28), and the trained protocol (blue curve) consisting of a piecewise linear interpolation with 10 learnable parameters. The initial and final stiffness values were $k_i = 1.0$ and $k_f = 5.0$, respectively, with a final time of $t_f = 1.0$ and a fixed center at $c = 0.0$. The inset depicts the evolution of the loss function during training, with the horizontal line representing the optimal work value, $\langle W \rangle_{\text{opt}} = 1.0565$.

through the first fully connected layer, which maps it to a space of dimension $n = 30$, followed by the application of the ReLU activation function. The output of this layer is fed into a second layer, where a Tanh activation is applied. A third layer follows, utilizing Leaky ReLU for handling small negative gradients. These deep layers have the same number of neurons $n = 30$. The fourth and final layer produces a scalar output using a modified Tanh to constrain the result within the range of k_i to k_f , see footnote.¹ The neural network was trained for 100 epochs with a learning rate of $\eta = 0.7$. The results are shown in Fig. 6. The left panel corresponds to the parameter set $k_i = 1.0$, $k_f = 5.0$, and $t_f = 0.1$ from which the optimal work is $\langle W \rangle_{\text{opt}} = 1.6937$ [Eq. (28)]. The final value of the loss function was $L_W = 1.6945$ representing a relative error of 0.04%. In the right panel of Fig. 6, the parameters were $k_i = 1.0$, $k_f = 2.0$, and $t_f = 1.0$, with an optimal work of $\langle W \rangle_{\text{opt}} = 0.4029$. The last value of the loss function was $L_W = 0.4053$ corresponding to a relative error of 0.60%. These results demonstrate that the network accurately reproduces the optimal stiffness protocol for both cases.

2. Optimizing work and equilibration time

Accelerating a system's equilibration through external control carries an energetic cost. Thus, a fundamental question is whether it is feasible to devise a control protocol that both accelerates equilibration and optimizes the work performed. This question holds significance not only from an experimental standpoint but also from a theoretical perspective. This

¹The Leaky ReLU function is stepwise that returns x if $x \geq 0$ and $-\epsilon x$ if $x < 0$, where ϵ is a positive small value, by default in PyTorch, $\epsilon = 10^{-2}$. If $\epsilon = 0$, then the function is called ReLU. The function Tanh is the usual hyperbolic tangent.

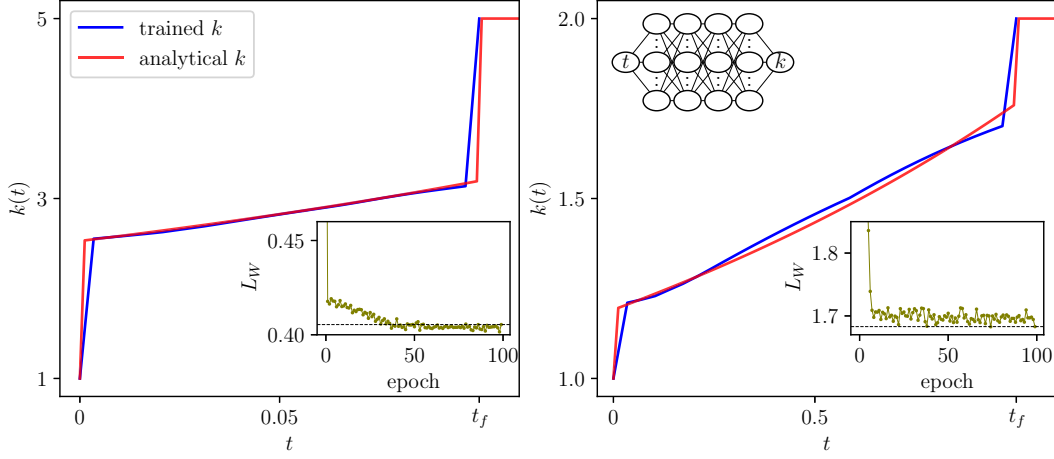


FIG. 6. Results after training a neural network to reproduce the stiffness that minimizes the work done on the system. The blue curve represents the trained model, while the red curve corresponds to the analytical solution given by Eq. (28). The insets display the evolution of the mean work (loss function) during training. The left panel corresponds to parameters $k_i = 1.0$, $k_f = 5.0$, and $t_f = 0.1$, whereas the right panel corresponds to $k_i = 1.0$, $k_f = 2.0$, and $t_f = 1.0$.

issue has been solved in Ref. [39] where the optimal control protocol is the same Eq. (28); however, the constant A now is given by

$$A = \frac{1}{t_f} \left(\frac{k_i}{k_f} - 1 \right), \quad (31)$$

and the fast thermal optimal work is

$$\langle W \rangle_{\text{opt}} = \frac{1}{2} \ln \left(\frac{k_f}{k_i} \right) + \frac{1}{k_i t_f} \left(\sqrt{\frac{k_i}{k_f}} - 1 \right)^2. \quad (32)$$

Observe that the quasistatic expression is reached if $t_f \rightarrow \infty$. See Ref. [39]. The change in the constant A is due to a modification of the boundary conditions that ensures that the final state is an equilibrium state, as required by rapid thermal equilibration. To replicate this protocol, we now require a loss function that encompasses the two conditions to be optimized. Consequently, we propose a loss function L_{FTW}

$$L_{\text{FTW}} = (1 - \alpha) L_{\text{char}} + \alpha L_W. \quad (33)$$

This is a convex linear combination of L_{char} and L_W of Eqs. (16) and (27). The blend parameter α works as a hyperparameter that is introduced by the user. The model used was $N_p = 10$ piecewise linear parametrization. The training was more challenging than the previous ones because the minimization of L_{FTW} represents a compromise between minimizing the work done on the system and reducing the distance to the target equilibrium state. Depending on the value of α one can give more importance to one of these objectives. The training was performed in batches of 30 epochs, with a variable learning rate of η in the range 15 000 to 30 000 and a variable blending α . We started with $\alpha = 0$ to let the algorithm find a fast thermal equilibration protocol, regardless of the work done. From there, we evaluated the average work L_W to be able to compare it with L_{char} and choose an appropriate blend α so that both contributions to L_{FTW} are weighted equally. This resulted in choosing α in the range from 10^{-5} to

10^{-4} for batches of 30 epochs and adjusting accordingly after each batch.

Finally, we obtained an acceptable optimal protocol, as depicted in Fig. 7. The theoretical value provided by Eq. (32) is $\langle W \rangle_{\text{opt}} = 5.4937$, whereas the simulation yields a value of $L_W = \langle W \rangle = 5.5030$. The relative error is 0.17%, and the final loss function L_{char} value was noted of order 10^{-6} , indicating that the equilibrium state has been reached. Note that the trained protocol $k(t)$ in Fig. 7 is smaller from the theoretical one in the last time steps. This shows the compromise mentioned earlier between optimizing the work [smaller values of $k(t)$] and reaching the equilibrium step at t_f which requires larger values of $k(t)$.

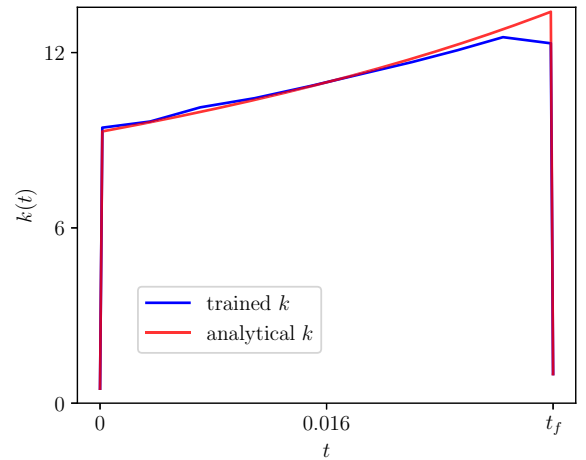


FIG. 7. Stiffness k as a function of time designed to accelerate equilibration and minimize work done on the system. The red curve represents the optimal protocol Eq. (28) with A given by Eq. (31), see Ref. [39]. Meanwhile, the blue curve illustrates the trained protocol. The parameters were $k_i = 0.5$, $k_f = 1.0$, and $t_f = 1/30$.

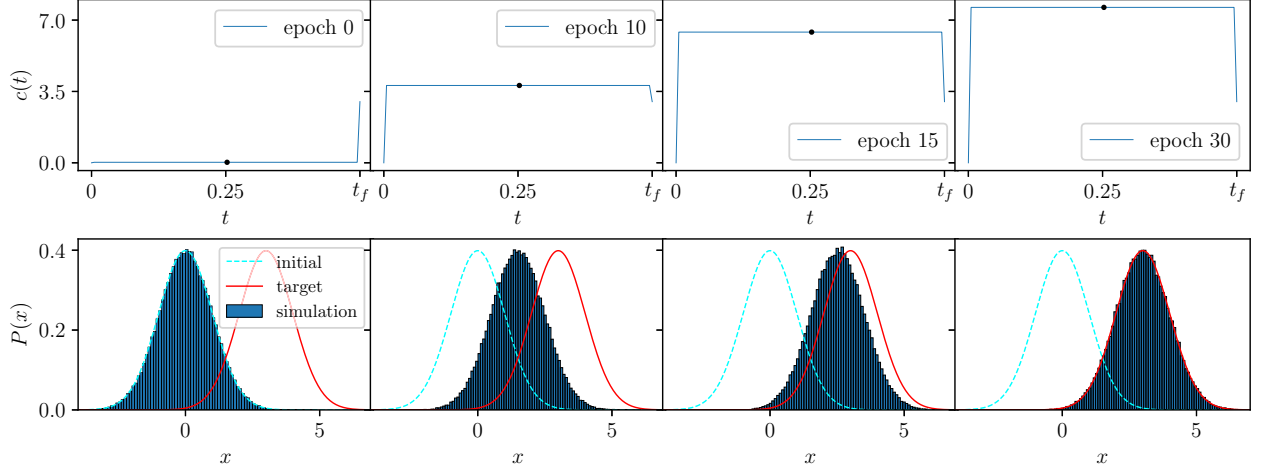


FIG. 8. Training of a single-parameter model to replicate the two-step protocol for the center for fast thermal equilibration, with $c_i = 0.0$, $c_f = 3.0$, and $t_f = 0.5$. The initial and target probability distributions are displayed, alongside a histogram generated from the model trained up to the despicted epoch.

C. The center of the harmonic potential as a control parameter

In the previous sections, we have focused on the case where the stiffness varies over time. However, the same protocol parametrization techniques apply when the stiffness is constant at $k = 1$, while the center position varies with time. To demonstrate that our machine learning approach is adaptable in this context as well, this section explores rapid thermal equilibration and work optimization by treating the center as a control parameter and employing a piecewise-linear parametrization.

The harmonic force in this case is

$$F(x, \lambda(t)) = -k(x - c(t)), \quad (34)$$

where $\lambda(t)$ is now playing the role of the center of the harmonic potential $c(t)$. Initially, we consider the case with a single learnable parameter $N_p = 1$, denoted as c_m . This scenario resembles a two-step protocol for stiffness but is now applied to the center. Let c_i and c_f be the initial and final values of the center, respectively. Then, the two-step protocol for the center is

$$c(t) = \begin{cases} c_i, & \text{if } t \leq 0, \\ c_m, & \text{if } 0 < t < t_f, \\ c_f, & \text{if } t \geq t_f. \end{cases} \quad (35)$$

For the force Eq. (34) and initial condition $P(x, 0) = e^{-k(x-c_i)^2/2} \sqrt{k/(2\pi)}$, the Fokker-Planck equation admits a solution

$$P(x, t) = \frac{1}{\sqrt{2\pi/k}} e^{-\frac{k}{2}(x - \langle x(t) \rangle)^2} \quad (36)$$

with

$$\langle x(t) \rangle = c_m + (c_i - c_m)e^{-kt} \quad (37)$$

for $0 < t < t_f$. The adjustable parameter c_m is determined assuming equilibrium at $t = t_f$, that is, the average position, $\langle x(t) \rangle$, is c_f at t_f . This gives rise to the value of

$$c_m = \frac{c_f - c_i e^{-kt_f}}{1 - e^{-kt_f}}. \quad (38)$$

If the duration of the protocol t_f is shorter than the natural relaxation time, we achieve a fast thermal equilibration.

Using a piecewise linear model with a single adjustable parameter $N_p = 1$ and training the model using the loss function defined in Eq. (16), we can compare the value of the adjustable parameter to the solution of Eq. (38) to evaluate the predictive accuracy of the algorithm. The training process was carried out with the following parameters: $c_i = 0$, $c_f = 3.0$, and $t_f = 0.5$ for 30 epochs with a learning rate of $\eta = 10$. For this set of parameters, the characteristic relaxation time $\tau = 1/k = 1.0$. We choose a protocol duration $t_f = 0.5$ smaller than the relaxation time. The results of training are depicted in Fig. 8. This figure shows the initial and final probability density functions after each epoch. After time t_f , the system reaches the required equilibrium state. During training, the loss function was approximately 10^{-2} at epoch 15, and the training continued until epoch 30, when the loss function decreased to approximately 10^{-5} . The trained value of c_m obtained through this process was $c_m = 7.6241$, compared to the expected value of $c_m = 7.6245$. This results in a deviation of approximately 0.004% from the expected value.

Now we consider the general case where the center is an arbitrary function of time. In this situation, no analytical solution exists in general. The center is parameterized using a piecewise linear function with $N_p = 10$ learnable parameters. This model assumes that $c(t)$ is a continuous function in $t = 0$ and $t = t_f$. The model is trained with a loss function that is a convex linear combination of L_{char} and a penalty term L_{gradc} . The term L_{gradc} penalizes strong variations in $c(t)$ and has the same structure as Eq. (23) for $k(t)$. The model underwent two training stages. The first stage, which lasted for 50 epochs, used a blend parameter of $\alpha = 10^{-3}$. The second stage, which lasted for 40 epochs, used $\alpha = 0$. Both stages used a learning rate of $\eta = 50$. The training process for selected epochs is illustrated in Fig. 9, which displays the probability density functions at various epochs. At epoch 50, the loss function was about 10^{-3} , and training continued until epoch 90, when the loss function reached the order of 10^{-6} , indicating that the system is in the final equilibrium state. As shown in Fig. 9,

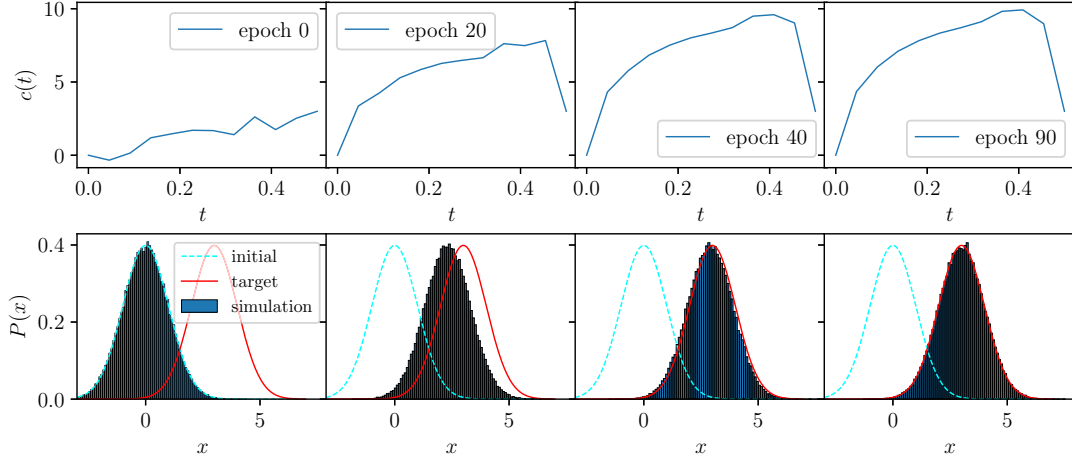


FIG. 9. Training for the center for fast thermal equilibration using a 10 parameter model and the loss function include the penalty term L_{gradc} . The initial (solid), target (dashed), and histogram of the trained model are displayed after some epochs. The parameters were $c_i = 0$, $c_f = 3.0$, and $t_f = 0.5$.

the model achieves high precision and the probability density function at the end aligns with the expected equilibrium distribution.

We now examine the scenario where the loss function is characterized by the average work, as defined in Eq. (27). As demonstrated in Ref. [38], the optimal protocol $c_0^*(t)$, and the optimal work $\langle W \rangle_{\text{opt}}$ are

$$c_0^*(t) = \frac{c_f(t+1)}{t_f+2}, \quad \langle W \rangle_{\text{opt}} = \frac{c_f^2}{t_f+2} \quad (39)$$

for $0 < t < t_f$. It is evident that the protocol $c_0^*(t)$ exhibits discontinuities in $t = 0.0$ and $t = t_f$. As indicated in Fig. 10, the precision in determining the optimal protocol is notably

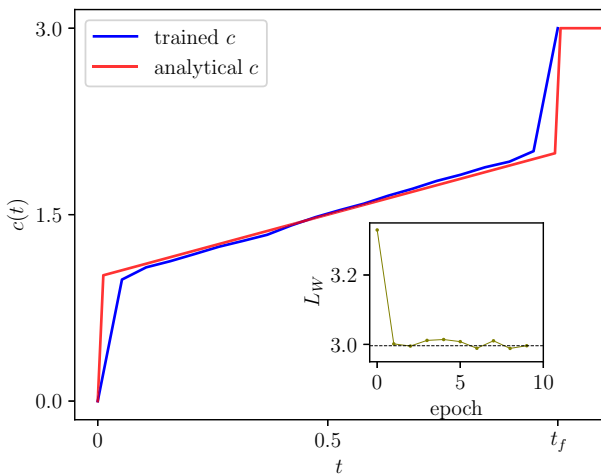


FIG. 10. Center position as a function of time to optimize the work done on the system. The red curve is the analytical solution Eq. (39), and the blue line is the trained protocol. The inset illustrates the loss function Eq. (27) during the training, while the dotted line indicates the optimal work, as in Eq. (39). The initial and final positions were $c_i = 0.0$ and $c_f = 3.0$, respectively. The final time was fixed at $t_f = 1.0$, and the stiffness was set to $k = 1.0$.

high in just 10 epochs and a learning rate of $\eta = 10$. For the set of parameters $c_i = 0$, $c_f = 3.0$, and $t_f = 1.0$, the theoretical value of the optimal work given by Eq. (39) is $\langle W \rangle_{\text{opt}} = 3.0$, while the result obtained through training after 10 epochs and a learning rate of $\eta = 10$ was $\langle W \rangle = 3.0060$. These results imply a relative error of 0.2%.

D. The center and stiffness of the harmonic potential as a control parameter

In this section, we consider the simultaneous adjustment of both the center and stiffness, which influence the system's dynamics through a nonlinear coupling described by the force in Eq. (34). As an initial example, we analyze a TSP for both parameters, where the stiffness k_m is determined by Eq. (22) and the center c_m by Eq. (38) with $k = k_m$.

The stiffness and center are modeled using a piecewise linear parametrization with $N_p = 1$ for each. Given that these parameters contribute differently to the dynamics—where k multiplies x while $c(t)$ does not—their gradients affect the optimization process differently. Consequently, different learning rates are required for each parameter.

For numerical simulations, we set $c_i = 1.0$, $k_i = 0.5$, $c_f = -1.0$, and $k_f = 1.0$. Based on these parameters, the simulations suggest a relaxation time of order $\tau = 4.0$. We aim to accelerate thermal equilibration in a final time $t_f = 0.1$. The theoretical values for k_m and c_m , obtained as solutions to Eqs. (22) and (38), are $k_m = 4.1931$ and $c_m = -4.8394$.

The model was trained for 100 epochs using learning rates $\eta_c = 10.0$ for the center and $\eta_k = (2/3)\eta_c$ for stiffness, resulting in a TSP solution with $k_m = 4.1218$ and $c_m = -4.9733$. The corresponding relative errors are 1.69% and 2.76%, respectively. The results presented in Fig. 11, confirm that the equilibrium state is successfully reached at t_f by the final epoch.

The more general case is where the stiffness and center are not constant in $[0, t_f]$. To address this scenario, we parametrize each variable using a piecewise linear model with $N_p = 10$, ensuring that the control variables remain continu-

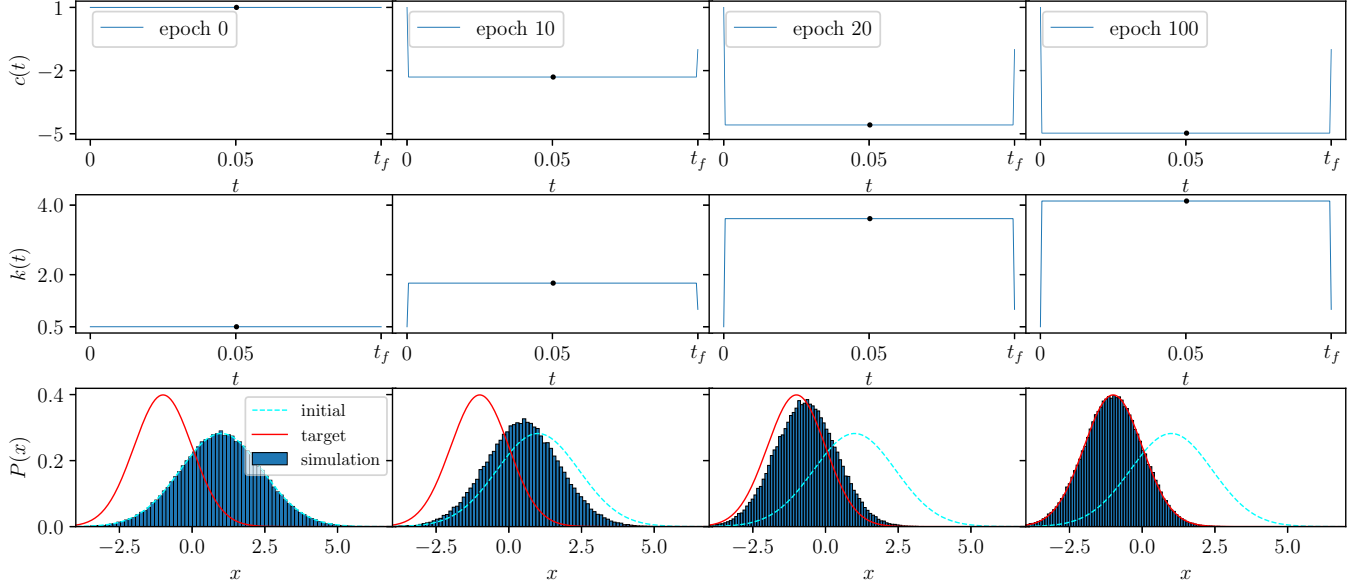


FIG. 11. Evolution of training of a model to reproduce the TSP for stiffness and center simultaneously. The initial (solid) and target (dashed) distributions correspond to equilibrium states, each described by Gaussian distributions with center c and variance $1/\sqrt{k}$. The histogram was generated with the trained model until the epoch shown. The parameters were $c_i = -c_f = 1.0$, $k_f = 1 = 2k_i$, and $t_f = 0.1$.

ous in $[0, t_f]$. Given that strong variations are expected at the beginning and end of the protocol, we introduce a penalty term in the loss function, $L_{\text{grad}k} + L_{\text{grad}c}$, to mitigate abrupt changes in k and c over the time interval $[0, t_f]$.

The model was initially trained for 100 epochs using the total loss function with a blend of $\alpha = 10^{-2}$, followed by an additional 10 epochs trained exclusively with L_{char} . Rep-

resentative epochs are depicted in Fig. 12, which illustrates that both k and c exhibit smooth variations and continuity. The final epoch confirms that the system successfully reaches equilibrium at $t = t_f$.

Through these examples, we have demonstrated that the algorithm successfully finds optimal protocols within the harmonic framework, effectively reproducing established an-

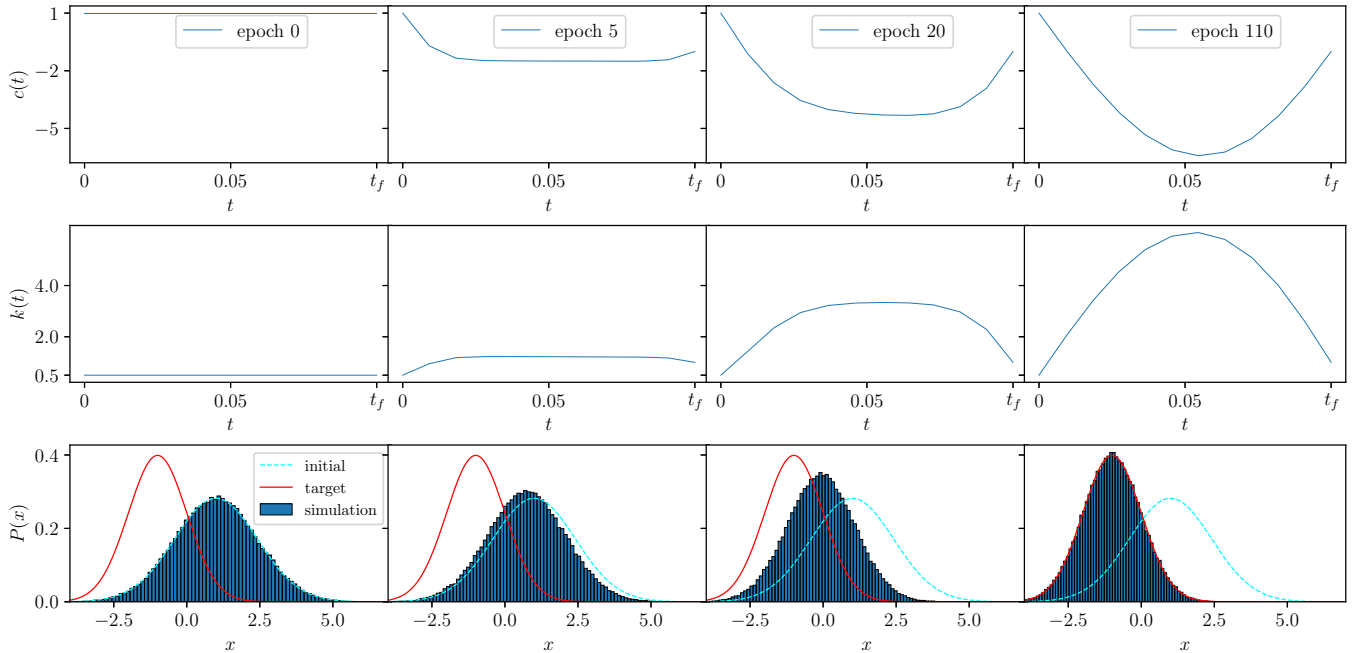


FIG. 12. The training evolution of a fast thermal equilibration model incorporating 10 parameters for stiffness and 10 parameters for center is shown. The loss function employed is a convex linear combination of L_{char} and $L_{\text{grad}k} + L_{\text{grad}c}$. The system is initially trained for 100 epochs using a blend parameter of 10^{-2} , followed by an additional 10 epochs with the loss function L_{char} to emphasize the fast thermal process. The parameter values were set as $k_f = 2k_i = 1.0$ and $c_f = -c_i = -1.0$, with a learning rate of $\eta_c = (3/2)\eta_k = 100$.

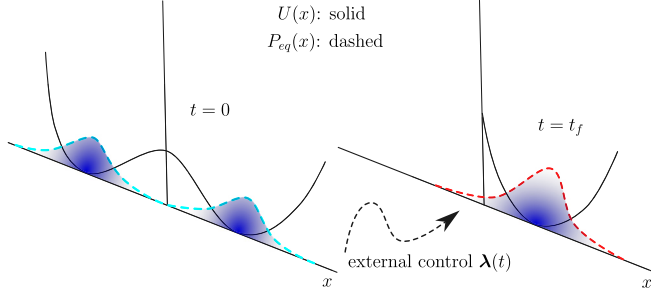


FIG. 13. Illustration of the externally controlled deformation of a fourth-degree potential at $t = 0$ into a parabolic potential at $t = t_f$. Multiple pathways $\lambda(t)$ exist to achieve this transformation.

alytical results. Furthermore, its applicability extends to the discovery of more complex protocols, as explored in the following section.

IV. BROWNIAN PARTICLE UNDER NONHARMONIC POTENTIALS

With minimal modifications, our piecewise linear parametrization model can be adapted to tackle more complex problems where analytical methods provide limited or no insight. As discussed in Ref. [38], solving optimization problems for nonharmonic potentials presents significant challenges, often rendering analytical solutions impractical and necessitating numerical techniques from optimal transport theory [28,30].

In this section, we demonstrate how our algorithm extends beyond the harmonic potential to address fast-thermalization problems. Specifically, we investigate the design of fast thermal protocols for transforming a fourth-degree polynomial potential energy at $t = 0$ into a quadratic potential at t_f . Given the importance of double-well potentials and their deformation into quadratic potentials in the context of memory erasure [40,41], we consider an initial potential with a double-well shape. The final state can be any confining potential; as an illustrative example, we select a quadratic potential obtained from the Taylor expansion of the initial potential around one of its minima (e.g., the right minimum). However, alternative final states can also be chosen. See Fig. 13 for details. Similarly to the harmonic case, there exist infinitely many possible paths connecting the initial and final states. Thus, our goal is to identify an external protocol that accelerates equilibration relative to the natural relaxation time.

We consider a time-dependent potential energy of the form:

$$U(x, \lambda(t)) = a_4(t)x^4 + a_3(t)x^3 + a_2(t)x^2 + a_1(t)x + a_0(t), \quad (40)$$

where the coefficients $\lambda = \{a_0, a_1, a_2, a_3, a_4\}$ are functions of time and serve as control parameters. A straightforward approach would be to parametrize each coefficient and train the model to minimize L_{char} . However, this method presents challenges. During training, the parameters are updated following Eq. (6), which can result in a repulsive potential energy, causing the particle to escape to infinity. This issue not only affects numerical simulations but also poses a fundamental obstacle

to the experimental realization of the protocol. This can be understood in the following way. The force act during the training is

$$F(x, \lambda(t)) = -(4a_4(t)x^3 + 3a_3(t)x^2 + 2a_2(t)x + a_1(t)). \quad (41)$$

If the algorithm updates the parameter a_4 to be negative for some time (or times), then the force tends to move the particle to the $|x| \rightarrow \infty$ directions, then the algorithm diverge. If the parameter a_4 is very close to zero by the positive numbers and the coefficient $a_3 > 0$, then the coefficient a_3 makes the main contribution to the force. Therefore, if the particle is located in $x < 0$, then it has a tendency to move to the left, that is, it moves away from the expected final position distribution, which in this case is around $x = 1.0$. This behavior leads to instabilities and convergence problems. Thus, we impose an additional constraint, given by $a_3(t) < 0$, a constraint motivated purely by the numerical convergence. In summary, our model included $a_4(t) > 0$ and $a_3(t) < 0$ in $t \in (0, t_f)$ as restrictions. To address this, the model must incorporate constraints that ensure that the potential remains confining and converge. This can be achieved either by explicitly enforcing confinement conditions within the model or by introducing a penalty term in the loss function to discourage nonphysical solutions. We used a model with the restrictions incorporated. To find the fast-thermalization protocol while ensuring confinement, we trained a model with two different loss functions. In both approaches, we modeled each coefficient with a piecewise linear parametrization with $N_p = 20$ parameters and including the restriction $a_4(t) > 0$ and $a_3(t) < 0$ in $(0, t_f)$. In addition, we chose the potential at $t = 0$ to have two symmetric minima at $x = -1.0$ and $x = 1.0$ with energy 0, a maximum at $x = 0.0$ with a separation barrier $U(0, 0) = 4.0$. At the end of the protocol, we selected the potential at $t = t_f$ to be a harmonic potential with minimum centered at $x = 1.0$ corresponding to the Taylor expansion to second order of the initial quartic potential around $x = 1.0$. To determine the system's relaxation time, we proceed as in the harmonic case, that is, the initial potential is abruptly switched to the final one (step protocol), and the time required for the system to reach equilibrium is observed. For this set states, the relaxation time was $\tau = 0.6$. Hence, we established the value of t_f as 0.06, making the protocol approximately 10 times faster than the relaxation time.

It is important to note that in this section we employed the Adam optimizer rather than stochastic gradient descent. This choice was motivated by the observation that the coefficients in the quartic potential play different roles in ensuring the boundedness of the potential. Consequently, it is essential to exercise precise control over the manner in which these parameters are updated during the training process. Adam proved to be more robust in this context because it utilizes adaptive learning rates and incorporates momentum, allowing for individualized parameter updates [42]. This adaptive mechanism is particularly beneficial when dealing with parameters that vary in importance, as it facilitates efficient convergence and helps mitigate the risk of stagnation during training.

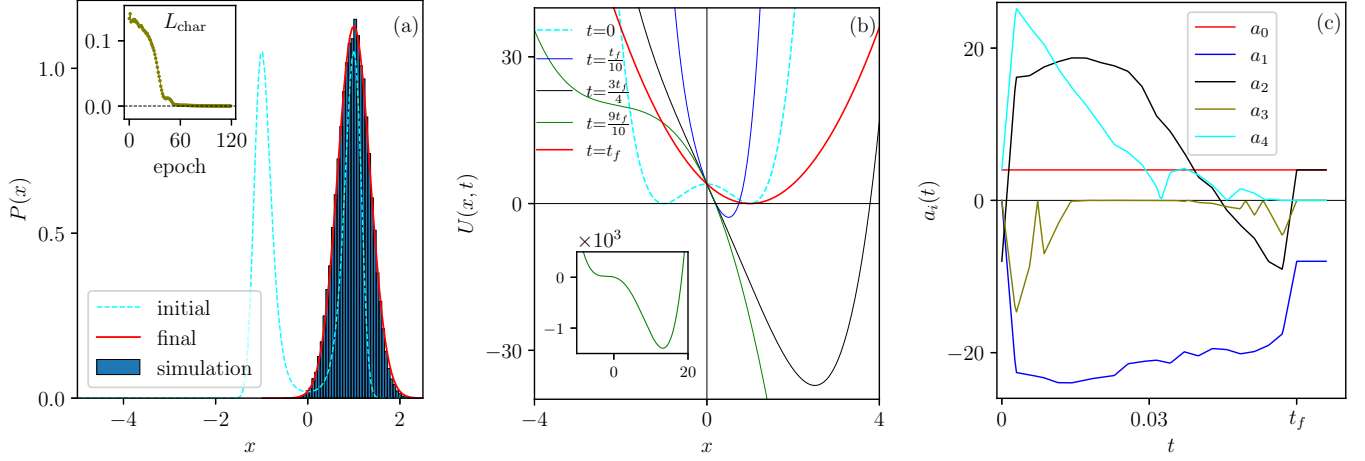


FIG. 14. Fast thermal protocol that transforms a quartic potential into a harmonic potential without smoothing the parameters changes. (a) Histogram of the trained model in the last epoch, i.e., epoch 120. The dotted line represents the initial quartic potential, while the solid line corresponds to the final harmonic potential. The inset shows the loss function as a function of epochs. (b) Some snapshots of the potential energy of the trained model. (c) Coefficients of the trained model as a function time.

This model was trained for 120 epochs with the loss function Eq. (16) and a learning rate of $\eta = 0.5$. The histogram of the trained model is shown in Fig. 14(a). At the end of the training process, the loss function reaches a value on the order of 10^{-4} [inset plot of Fig. 14(a)], which means that the system has reached equilibrium. As illustrated in Fig. 14(b), the potential evolves to significantly higher values compared to its initial state, as highlighted in the inset. This increase in $|U|$ reflects the high probability that the particle will be found in these regions, with the external potential effectively guiding it toward the final equilibrium configuration. Despite the fact that this protocol is fast thermal, the Fig. 14(c) shows abrupt changes in the coefficients, a feature that is not desirable from the experimental point of view.

This behavior mirrors that observed in the harmonic case. Following the same ideas as those for the harmonic case, we introduced a penalty term in the loss function to discourage large variations in the coefficients. The training process was divided into two stages. In the first stage (up to epoch 100), we employed the loss function:

$$L = (1 - \alpha)L_{\text{char}} + \alpha L_{\text{grada}}, \quad (42)$$

$$L_{\text{grada}} = \sum_{i=0}^4 \sum_{n=0}^{N_p-1} (a_i(t_{n+1}) - a_i(t_n))^2 \quad (43)$$

where the blend parameter was set to $\alpha = 10^{-3}$ and the learning rate to $\eta = 0.5$. At the end of this stage, the loss function was of order 10^{-3} . In the second stage, the model was further trained for 50 epochs using only the loss function L_{char} (i.e., setting $\alpha = 0$) to focus entirely on optimizing fast-thermal equilibration, and the final value of the loss function was 10^{-5} .

Figure 15 resumes the results for this case. Figure 15(a) presents the histogram obtained with the trained model, and the inset describes the evolution of the loss function. In Fig. 15(b) selected snapshots of the optimized potential were plotted, but remarkably, the variations in U are two orders

of magnitude less than in the case without the penalty term [see the inset of Fig. 15(c)]. Figure 15(c) plots the temporal evolution of the trained potential coefficients, demonstrating the expected smoothing effect introduced by the penalty term. The gradual variation of the coefficients ensures a more stable and controlled evolution of the potential, which is particularly advantageous for experimental implementations, where abrupt parameter changes can be challenging to realize.

Both implementations successfully achieve the desired transformation, although the penalty-based approach offers a more feasible solution. These findings highlight the flexibility of our framework in adapting to different physical constraints, paving the way for practical implementations in experimental settings.

V. CONCLUSIONS

In this study, we demonstrated how to apply machine learning tools to find control protocols that optimize conditions such as mean work or protocols that achieve equilibrium faster than the relaxation time, or both conditions taken simultaneously. From the practical point of view, we have shown the flexibility that this numerical approach offers. It is possible to parametrize protocols in different ways or include restrictions in the models that achieve a specific goal. These restrictions cannot be arbitrary but motivated from physical reasons. Otherwise, a machine learning algorithm could find solutions that lack physical meaning.

We reproduced the well-established results for the stiffness of the harmonic oscillator in the overdamped limit. These results have been reproduced using different model parametrizations to recall the flexibility of our approach. In addition, we showed that our method is also suitable for dealing with the center of the harmonic potential, indicating independence of the parameters used to perform the control on the stochastic system. Furthermore, we studied nonharmonic potential, and we concluded that our approach is perfectly well adapted to more complex potentials. This is one of the

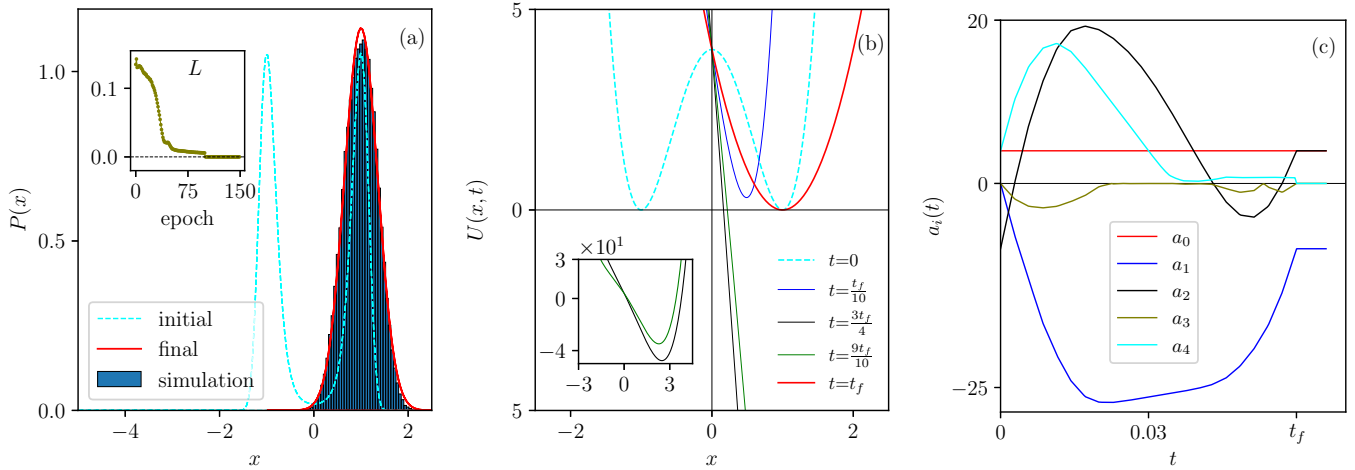


FIG. 15. Fast thermal protocol that transforms a quartic potential into a harmonic potential with smoothing the parameters changes. (a) Histogram of the trained model in the last epoch, i.e., epoch 120. The dotted line represents the initial quartic potential, while the solid line corresponds to the final harmonic potential. The inset shows the loss function as a function of epochs. (b) Some snapshots of the potential energy of the trained model. (c) Coefficients of the trained model as a function time.

most important results of this work. The scalability and adaptability of machine learning for stochastic systems offer a new numerical tool to deal with the complexities of stochastic control theory and stochastic thermodynamics.

While our approach offers several advantages, it also has certain limitations. Perhaps the most significant challenge is extending the framework to systems with multiple interacting particles. This requires solving coupled stochastic differential equations, often with different external control parameters, where analytical methods are virtually nonexistent. In addition, our approach is currently limited to the overdamped regime, and the inclusion of inertia represents some challenges [23,30,32,43]. Developing generalized algorithms capable of handling interactions and the underdamped regime remains an important direction for future work.

Another limitation is the lack of direct physical interpretation for some parameters, particularly those emerging from the neural network approach. The internal parameters of the network do not always correspond to physically meaningful quantities, which can make the results less intuitive. Additionally, there is considerable flexibility in selecting functions such as L_{char} and the penalty terms, which introduces an element of arbitrariness.

Despite these challenges, our results provide a foundation for exploring how machine learning techniques can be effectively applied to stochastic control dynamics and ther-

modynamics. Future work in this direction could lead to more refined and physically interpretable models, further bridging the gap between machine learning and statistical physics.

ACKNOWLEDGMENTS

We gratefully acknowledge the support of Universidad de los Andes, Facultad de Ciencias, Projects No. INV-2025-205-3271 and No. INV-2023-176-2951, and its High Performance Computing center (HPC). We acknowledge support from ECOS Nord Project No. C24P01 and Minciencias, Patrimonio Autónomo Fondo Nacional de Financiamiento para la Ciencia, la Tecnología y la Innovación, Francisco José de Caldas. We thank the Deutscher Akademischer Austauschdienst (DAAD) for financial support. We also thank Sabine Klapp and her research group for inspiring discussions and hospitality at Technical University Berlin, Germany.

DATA AVAILABILITY

The data that support the findings of this article are openly available [44]. The code we developed in this work is open source and it is available [44] as a python library which can be used to reproduce our results, train new protocols, and extend it for more complex potentials. The documentation of the code can be found in Ref. [45].

- [1] J. Bechhoefer, *Control Theory for Physicists* (Cambridge University Press, Cambridge, UK, 2021).
- [2] G. M. Wang, E. M. Sevick, E. Mittag, D. J. Searles, and D. J. Evans, *Phys. Rev. Lett.* **89**, 050601 (2002).
- [3] C. Gonzalez-Ballester, M. Aspelmeyer, L. Novotny, R. Quidant, and O. Romero-Isart, *Science* **374**, eabg3027 (2021).
- [4] D. Raynal, T. de Guillebon, D. Guéry-Odelin, E. Trizac, J.-S. Lauret, and L. Rondin, *Phys. Rev. Lett.* **131**, 087101 (2023).
- [5] I. Martínez, A. Petrosyan, D. Guéry-Odelin, E. Trizac, and S. Ciliberto, *Nat. Phys.* **12**, 843 (2016).
- [6] G. Li, H. T. Quan, and Z. C. Tu, *Phys. Rev. E* **96**, 012144 (2017).
- [7] D. Guéry-Odelin, C. Jarzynski, C. A. Plata, A. Prados, and E. Trizac, *Rep. Prog. Phys.* **86**, 035902 (2023).
- [8] S. Dago, B. Besga, R. Mothe, D. Guéry-Odelin, E. Trizac, A. Petrosyan, L. Bellon, and S. Ciliberto, *SciPost Phys.* **9**, 064 (2020).

- [9] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, *J. Mach. Learn. Res.* **18**, 1 (2018).
- [10] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, in *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research Vol. 37, edited by F. Bach and D. Blei (PMLR, Lille, France, 2015), pp. 2256–2265.
- [11] J. Ho, A. Jain, and P. Abbeel, in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)* (2020) [arXiv:2006.11239](https://arxiv.org/abs/2006.11239).
- [12] M. C. Engel, J. A. Smith, and M. P. Brenner, *Phys. Rev. X* **13**, 041032 (2023).
- [13] C. Casert and S. Whitlam, *Nat. Commun.* **15**, 9128 (2024).
- [14] A. Karpathy, The spelled-out intro to neural networks and back-propagation: Building micrograd, <https://www.youtube.com/watch?v=VMj-3S1tku0>.
- [15] Pytorch, <https://pytorch.org/>.
- [16] Tensorflow, <https://www.tensorflow.org/>.
- [17] Jax: High-performance array computing, <https://jax.readthedocs.io/>.
- [18] K. Hornik, M. Stinchcombe, and H. White, *Neural Networks* **2**, 359 (1989).
- [19] S. Whitlam and I. Tamblyn, *Phys. Rev. E* **101**, 052604 (2020).
- [20] C. P. Goodrich, E. M. King, S. S. Schoenholz, E. D. Cubuk, and M. P. Brenner, *Proc. Natl. Acad. Sci. USA* **118**, e2024083118 (2021).
- [21] S. Whitlam, *Phys. Rev. X* **13**, 021005 (2023).
- [22] S. Whitlam, *Phys. Rev. E* **108**, 044138 (2023).
- [23] N. Barros, S. Whitlam, S. Ciliberto, and L. Bellon, *Phys. Rev. E* **111**, 044114 (2025).
- [24] S. A. M. Loos, S. Monter, F. Ginot, and C. Bechinger, *Phys. Rev. X* **14**, 021032 (2024).
- [25] G. Biroli and M. Mézard, *J. Stat. Mech.* (2023) 093402.
- [26] G. Biroli, T. Bonnaire, V. de Bortoli, and M. Mézard, *Nat. Commun.* **15**, 9957 (2024).
- [27] S. Whitlam, V. Selin, I. Benlolo, C. Casert, and I. Tamblyn, *Mach. Learn.: Sci. Technol.* **3**, 045026 (2022).
- [28] E. Aurell, C. Mejía-Monasterio, and P. Muratore-Ginanneschi, *Phys. Rev. Lett.* **106**, 250601 (2011).
- [29] C. Villani, *Topics in Optimal Transportation*, Graduate Studies in Mathematics (American Mathematical Society, Providence, RI, 2003).
- [30] J. Sanders, M. Baldovin, and P. Muratore-Ginanneschi, *Phys. Rev. E* **111**, 034127 (2025).
- [31] J. Klinger and G. M. Rotskoff, *Phys. Rev. E* **111**, 014114 (2025).
- [32] P. Muratore-Ginanneschi and K. Schwieger, *Phys. Rev. E* **90**, 060102(R) (2014).
- [33] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, Neural ordinary differential equations, [arXiv:1806.07366](https://arxiv.org/abs/1806.07366).
- [34] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes* (Interscience Publishers, Geneva, 1962).
- [35] X. Li, T.-K. L. Wong, R. T. Q. Chen, and D. Duvenaud, Scalable gradients for stochastic differential equations, [arXiv:2001.01328](https://arxiv.org/abs/2001.01328).
- [36] D. Rengifo and G. Téllez, *Phys. Rev. E* **109**, 014145 (2024).
- [37] L. Peliti and S. Pigolotti, *Stochastic Thermodynamics: An Introduction* (Princeton University Press, Princeton, NJ, 2021).
- [38] T. Schmiedl and U. Seifert, *Phys. Rev. Lett.* **98**, 108301 (2007).
- [39] T. Schmiedl and U. Seifert, *Europhys. Lett.* **81**, 20003 (2008).
- [40] K. Proesmans, J. Ehrich, and J. Bechhoefer, *Phys. Rev. Lett.* **125**, 100602 (2020).
- [41] S. Dago, J. Pereda, N. Barros, S. Ciliberto, and L. Bellon, *Phys. Rev. Lett.* **126**, 170601 (2021).
- [42] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, [arXiv:1412.6980](https://arxiv.org/abs/1412.6980).
- [43] A. Gomez-Marin, T. Schmiedl, and U. Seifert, *J. Chem. Phys.* **129**, 024114 (2008).
- [44] Langesim optim library, https://github.com/GabrielTellez/langesim_optim.
- [45] Langesim optim documentation, https://gabrieltellez.github.io/langesim_optim.