Thermodynamic AI and Thermodynamic Linear Algebra

Anonymous Author(s)
Affiliation
Address
`email`

Abstract

Many Artificial Intelligence (AI) algorithms are inspired by physics and employ stochastic fluctuations, such as generative diffusion models, Bayesian neural networks, and Monte Carlo inference. These algorithms are currently run on digital hardware, ultimately limiting their scalability and overall potential. Here, we propose a novel computing device, called Thermodynamic AI hardware, that could accelerate such algorithms. Thermodynamic AI hardware can be viewed as a novel form of computing, since it uses novel fundamental building blocks, called stochastic units (s-units), which naturally evolve over time via stochastic trajectories. In addition to these s-units, Thermodynamic AI hardware employs a Maxwell's demon device that guides the system to produce non-trivial states. We provide a few simple physical architectures for building these devices, such as RC electrical circuits. Moreover, we show that this same hardware can be used to accelerate various linear algebra primitives. We present simple thermodynamic algorithms for (1) solving linear systems of equations, (2) computing matrix inverses, (3) computing matrix determinants, and (4) solving Lyapunov equations. Under reasonable assumptions, we rigorously establish asymptotic speedups for our algorithms, relative to digital methods, that scale linearly in dimension. Numerical simulations also suggest a speedup is achievable in practical scenarios.

1 Introduction

With recent breakthroughs from text-to-image to large language models, AI progress has exceeded the expectations of even many optimists. Much of this progress has happened through software and algorithmic advances. This includes the scaling up of deep neural-network architectures, which in turn was enabled by a hardware "fluke" for accelerating matrix-vector multiplications (MVMs) [28]. Namely, this has been driven by parallelized digital hardware such as graphical processing units (GPUs) and field-programmable gate arrays (FPGAs) [37]. More recently, analog hardware has been developed to reduce the power consumption in performing MVMs [18, 57], and there is a rich history of analog hardware for neural networks [8, 48, 53].

However, there may await another revolution in scaling up AI through fundamentally distinct, domain-specific hardware. This viewpoint has amassed increasing popularity, with initial strides where algorithm and hardware are considered inseparable [25]. The next revolution in AI hardware may involve connecting AI to physics, i.e., identifying the physical basis of intelligence. In support of this view, some of the most successful AI algorithms, such as generative diffusion models [52], time-series analysis with neural stochastic differential equations (SDEs) [36, 41] and Bayesian neural networks [23, 58], are inspired by physics and often employ stochastic fluctuations. This suggests that thermodynamic fluctuations could be a key ingredient for building AI. Instead

Submitted to the First Workshop on Machine Learning with New Compute Paradigms at NeurIPS (MLNPCP 2023). Do not distribute.

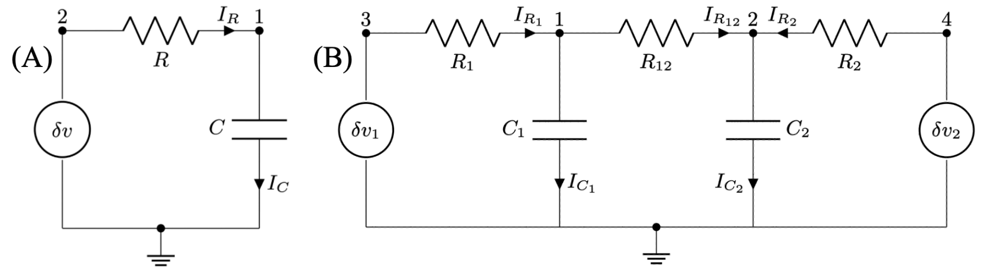

Figure 1: **Physical realization of s-modes.** (A) Circuit diagram of a possible physical realization of an s-mode, consisting of a noisy resistor and a capacitor. (B) Circuit diagram of a possible means to couple s-modes, using a coupling resistor.

of simulating such fluctuations with standard digital hardware, we propose a hardware paradigm that has thermodynamic fluctuations already built in as a fundamental building block, which we call Thermodynamic AI.

We note that Thermodynamic AI can be viewed as a sub-field of Thermodynamic Computing. The latter was discussed broadly in a workshop report [16], and was further explored with thermodynamic neural networks [31, 32], thermodynamic neuromorphic systems [20, 22, 21], probabilistic bit computers [10, 35, 1], work production [9], and other thermodynamic perspectives on learning [3, 51, 24, 4]. We use the term Thermodynamic AI to refer to a thermodynamic viewpoint on hardware for modern AI applications.

2 Fundamental building blocks

	classical	quantum	thermodynamic
discrete	bit	qubit	s-bit
continuous	mode	qumode	s-mode

Let us now discuss the fundamental building blocks of thermodynamic AI hardware. As the name suggests, a thermodynamic system is inherently dynamic in nature. Therefore, the fundamental building blocks should also be dynamic. This is contrast to classical bits or qubits, where the state of the system ideally remains fixed unless it is actively changed by gates. The thermodynamic building block should passively and naturally evolve over time, even without the application of gates. But what dynamical process should it follow? A reasonable proposal is a stochastic Markov process. Naturally this should be continuous in time, since no time point is more special than any other time point. Hence, the discrete building block, which we call an s-bit, would follow a continuous-time Markov chain (CTMC). Here the "s" in s-bit stands for stochastic. For the continuous building block, which we call an s-mode, the natural analog would be a Brownian motion (also known as a Wiener process). We use s-unit as a generic term to encompass both s-bits and s-modes. We note that stochastic building blocks were also considered by Mansinghka et al. [42, 43] using digital logic, which is different from our (analog) approach.

3 Physical realizations of stochastic units

While additional details on s-bits are given in Ref. [15], we focus our attention on s-modes, since continuous variables are particularly relevant to AI applications. An s-mode represents a continuous stochastic variable whose dynamics are governed by drift, diffusion, or other physical processes (akin to a Brownian particle). At the heart of any physical implementation of such a variable will be a source of stochasticity. A natural starting point for implementing thermodynamic AI hardware is analog electrical circuits, as these circuits have inherent fluctuations that could be harnessed for computation.

The most ubiquitous source of noise in electrical circuits is thermal noise [29]. Thermal noise, also called Johnson-Nyquist noise, comes from the random thermal agitation of the charge carriers in a conductor, resulting in fluctuations in voltage or current inside the conductor. Thermal noise is

Gaussian and has a flat frequency spectrum (white noise) with fluctuations in the voltage of standard deviation $v_{\mathrm{tn}} = \sqrt{4k_B T R \Delta f}$. Another type of electrical noise is shot noise [29], which arises from the discrete nature of charge carriers and from the fact that the probability of a charge carrier crossing a point in a conductor at any time is random. Regardless of the physical source, variable-gain amplifiers can allow one to independently control the amplitude of the fluctuations.

The s-mode can be represented through the dynamics of any degree of freedom of an electrical circuit. If we chose the voltage on a particular node in the circuit as our degree of freedom of choice, a simple stochastic voltage noise source plays the role of the s-mode. This can be realized by using a noisy resistor at non-zero temperature. Figure 1(A) shows the typical equivalent noise model for a noisy resistor composed of a stochastic voltage noise source, $\delta v(t)$, in series with an ideal (non-noisy) resistor of resistance R. The inherent terminal capacitance, C, of the resistor is also added to the equivalent resistor model. The voltage on node 1 (labeled simply as $v(t)$ here) is the state variable, whose dynamics obey:

$$-\dot{v}(t) = (v(t) + \delta v(t))/RC. \tag{1}$$

This stochastic differential equation (SDE) comprises a drift term proportional to $v(t)$ and a diffusion or stochastic term proportional to $\delta v(t)$.

When building systems of many s-modes, one will wish to couple them to express correlations and geometric constraints. Again, the medium of analog electrical circuits presents a natural option for the coupling of s-modes. As an example, two s-modes could be coupled through a resistor, as pictured in Fig. 1(B). The coupled s-modes, represented by the voltage on nodes 1 and 2, are then coupled through their drift terms as

$$-\dot{\mathbf{v}} = \mathbf{C}^{-1}\left(\mathbf{J}\mathbf{v} + \mathbf{R}^{-1}\delta\mathbf{v}\right), \tag{2}$$

where $\dot{\mathbf{v}} \equiv \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{v}$ and

$$\mathbf{v} \equiv \begin{bmatrix} v_1 \\ v_2 \end{bmatrix},\ \mathbf{C} \equiv \begin{bmatrix} C_1 & 0 \\ 0 & C_2 \end{bmatrix},\ \mathbf{R} \equiv \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix},\ \mathbf{J} \equiv \begin{bmatrix} \frac{1}{R_1} + \frac{1}{R_{12}} & -\frac{1}{R_{12}} \\ -\frac{1}{R_{12}} & \frac{1}{R_2} + \frac{1}{R_{12}} \end{bmatrix},\ \delta\mathbf{v} \equiv \begin{bmatrix} \delta v_1 \\ \delta v_2 \end{bmatrix}.$$

Here we introduced the self-resistance matrix $\mathbf{R}$, the capacitance matrix $\mathbf{C}$, and the conductance matrix $\mathbf{J}$. Capacitive coupling is an alternative means to couple s-modes [15].

4 Maxwell's Demon

Maxwell's Demon is a thermodynamic concept that is similar to refrigeration, as it allows a system's entropy to be reduced over time by interacting with an external system. Here, the demon acts as an intelligent observer who regularly gathers data from (i.e., measures) the system, and based on the gathered information, the demon performs some action on the system. We argue that a Maxwell Demon is both: (1) Essential to the success of Thermodynamic AI systems due to the complex entropy dynamics required for AI applications, and (2) Quite straightforward to implement in practice for several different hardware architectures.

Regarding the first point, AI applications like Bayesian inference aim to approximate a posterior distribution, and it is known that such posteriors can be extremely complicated and multi-modal [33]. Similarly, generative modeling is intended to handle arbitrary data distributions. Hence, producing only Gaussian distributions, as an isolated s-mode system would do, will not suffice for these applications. Regarding the second point, one can either use digital or analog hardware to implement a Maxwell's Demon. A digital Maxwell's Demon can correspond to a neural network that is stored on a digital central processing unit, which then communicates back-and-forth with the stochastic unit system via analog / digital signal interconversion. An analog Maxwell's Demon could allow one to integrate it more closely to the rest of the thermodynamic hardware. Moreover, this could allow one to avoid interconverting signals. Various analog approaches are discussed in Ref. [15].

5 Thermodynamic Noise Robustness

While analog and quantum computing view noise as a nuisance or roadblock, Thermodynamic AI views noise as essential. The electrical circuits in Thermodynamic AI will have unintentional, uncontrollable noise. But from a mathematical perspective, this noise can be combined with whatever noise sources that one intentionally engineers. Under reasonable assumptions, this leads to the same mathematical form of the the differential equations governing the time evolution, i.e., unintentional noise preserves the mathematical framework. From a practical perspective, this unintentional noise can either be calibrated and compensated for, or it can be overwhelmed by the intentionally injected noise (and hence neglected). In addition, other sources of errors, such as parasitic couplings and component mis-specifications, can be compensated for in the training of the Maxwell's Demon, i.e., successful training will automatically imply good performance despite these error sources.

6 AI algorithms suited for Thermodynamic AI Hardware

Some modern AI algorithms that exploit stochasticity include: (1) Generative diffusion models, (2) Bayesian neural networks, (3) Monte Carlo inference, (4) Annealing, and (5) Time series forecasting. Each of these applications is elaborated in more detail in the Supplementary Material.

Through careful consideration, we manage to formulate a mathematical framework that encompasses all of the aforementioned algorithms as special cases. We say that these algorithms belong to a class called *Thermodynamic AI algorithms*. At a conceptual level, we can define Thermodynamic AI algorithms as those consisting of at least two subroutines:

1. A subroutine in which a stochastic differential equation (SDE) is evolved over time.
2. A subroutine in which a Maxwell's demon (see Sec. 4 for elaboration) observes the state variable in the SDE and applies a drift term in response.

At the mathematical level, we propose that Thermodynamic AI algorithms are ones that simulate or implement the following set of equations (or some subset of them):

$$d\mathbf{p} = [\mathbf{f} - BM^{-1}\mathbf{p}]\,dt + D\,d\mathbf{w} \tag{3}$$

$$d\mathbf{x} = M^{-1}\mathbf{p}\,dt \tag{4}$$

$$\mathbf{f} = -\nabla_{\mathbf{x}} U_\theta \tag{5}$$

One can see that these correspond to Newton's laws of motion, with the addition of diffusion and friction. In these equations, $\mathbf{p}$, $\mathbf{x}$, and $\mathbf{f}$ respectively are the momentum, position, and force. The matrices M, D, and B are hyperparameters, with M being the mass matrix and D being the diffusion matrix. The $d\mathbf{w}$ term is a Wiener process. Finally, U_θ is a (trainable) potential energy function. Typically, much of the application-specific information, regarding the task to be solved, is encoded in the potential energy function U_θ.

Note that the unification of various AI algorithms under the same framework (i.e., the above equations) is crucial to developing a hardware paradigm that is broadly applicable to many applications. We encourage the reader to see the Supplementary Material for details on specific applications.

7 Thermodynamic Linear Algebra

Basic linear algebra primitives are at the core of many algorithms in engineering and science. They are also a common subroutine of many artificial intelligence (AI) algorithms, and account for a substantial portion of the time and energy costs in some cases. Here we discuss how Thermodynamic AI hardware can be used for accelerating such linear algebra primitives.

7.1 Solving Linear Systems

We focus here on the case of solving linear systems, while details about other primitives like matrix inversion and matrix determinants can be found in the Supplementary Material (see also Ref. [2]). The celebrated linear systems problem is to find $x \in \mathbb{R}^d$ such that $Ax = b$ given some invertible matrix $A \in \mathbb{R}^{d\times d}$ and nonzero $b \in \mathbb{R}^d$. We assume that A is symmetric and positive definite (SPD) since the non-SPD case can be reformulated as an SPD linear system by considering the equation $A^\intercal Ax = A^\intercal b$.

Now let us connect this problem to thermodynamics. We consider a macroscopic device with d degrees of freedom, described by classical physics. Suppose the device has potential energy function:

$$V(x) = (1/2)x^\intercal Ax - b^\intercal x, \tag{6}$$

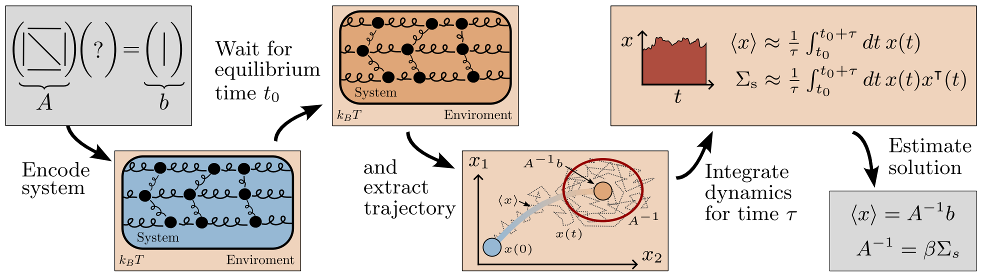
A
b
Wait for equilibrium time t_0
System
k_BT
Enviroment
Encode system
and extract trajectory
x_1
x_2
$A^{-1}b$
$\langle x \rangle$
$x(t)$
$x(0)$
A^{-1}
x
t
$\langle x \rangle \approx \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dt\, x(t)$
$\Sigma_s \approx \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dt\, x(t)x^\intercal(t)$
Integrate dynamics for time τ
Estimate solution
$\langle x \rangle = A^{-1}b$
$A^{-1} = \beta\Sigma_s$

Figure 2: **Diagram of our thermodynamic algorithm for solving linear systems and inverse estimation.** The system of linear equations, or the matrix A, is encoded into the thermodynamic hardware, the system is then allowed to evolve until the stationary distribution has been reached, when the trajectory is then integrated to estimate the sample mean or covariance. This gives estimates of the solution of the linear system or the inverse of A respectively.

where $A \in \text{SPD}_d(\mathbb{R})$. Note that this is a quadratic potential that can be physically realized with a system of harmonic oscillators, where the coupling between the oscillators is determined by the matrix A, and the b vector describes a constant force on each individual oscillator. (We remark that while Figure 2 depicts mechanical oscillators, from a practical perspective, one can build the device from electrical oscillators such as RLC circuits, similar to the s-modes described above.)

Suppose that we allow this device to come to thermal equilibrium with its environment, whose inverse temperature is $\beta = 1/k_BT$. At thermal equilibrium, the Boltzmann distribution describes the probability for the oscillators to have a given spatial coordinate: $f(x) \propto \exp(-\beta V(x))$. Because $V(x)$ is a quadratic form, $f(x)$ corresponds to a multivariate Gaussian distribution. Thus at thermal equilibrium, the spatial coordinate x is a Gaussian random variable

$$x \sim \mathcal{N}[A^{-1}b, \beta^{-1}A^{-1}]. \tag{7}$$

The key observation is that the unique minimum of $V(x)$ occurs where $Ax - b = 0$, which also corresponds to the unique maximum of $f(x)$. For a Gaussian distribution, the maximum of $f(x)$ is also the first moment $\langle x \rangle$. Thus, we have that, at thermal equilibrium, the first moment is the solution to the linear system of equations: $\langle x \rangle = A^{-1}b$. From this analysis, we can construct a simple thermodynamic protocol for solving linear systems, which is depicted in Figure 2. Namely, the protocol involves realizing the potential in Eq. (6), waiting for the system to come to equilibrium, and then sampling x to estimate the mean $\langle x \rangle$ of the distribution.

7.2 Algorithmic Scaling

In Table 1, we summarize the asymptotic scaling results for our thermodynamic algorithms as compared to the best state-of-the-art (SOTA) digital methods for dense symmetric positive-definite matrices. As one can see from Table 1, an asymptotic speedup is predicted for our thermodynamic algorithms relative to the digital SOTA algorithms. Specifically, a speedup that is linear in d is expected for each of the linear algebraic primitives (ignoring a possible dependence of κ on d).

Our numerical simulations (which are based on a detailed timing model [2]) corroborate our analytical scaling results and also provide evidence of the fast convergence of these primitives with the wall-clock time, with the speedup relative to digital methods getting more pronounced with increasing dimension and condition number. This is illustrated in Fig. 3.

8 Conclusions

There are several take-home messages that we would like to share with the reader:

- There is an opportunity for a new physics-based computing paradigm where the hardware is stochastic by design. We identified the key ingredients of that hardware paradigm as a stochastic-unit (s-unit) system coupled to a Maxwell's demon device.
- AI applications stand to benefit most from this hardware since many such applications are inherently stochastic.

Problem	Digital SOTA	Overdamped TA	Underdamped TA
Linear System	$O(\min\{d^\omega, d^2\sqrt{\kappa}\})$	$O(d\kappa^2\varepsilon^{-2})$	$O(d\sqrt{\kappa}\varepsilon^{-2})$
Matrix Inverse	$O(d^\omega)$	$O(d^2\kappa\varepsilon^{-2})$	$O(d^2\kappa\varepsilon^{-2})$
Lyapunov Equation	$O(d^3)$	$O(d^2\kappa\varepsilon^{-2})$	$O(d^2\kappa\varepsilon^{-2})$
Matrix Determinant	$O(d^3)$	$O(d\kappa\ln(\kappa)^3\varepsilon^{-2})$	$O(d\ln(\kappa)^3\varepsilon^{-2})$

Table 1: **Comparison of asymptotic complexities of linear algebra algorithms.** Here, d is the matrix dimension, κ is the condition number, and ϵ is the error. For our thermodynamic algorithms (TAs), the complexity depends on the dynamical regime, i.e., whether the dynamics are overdamped and underdamped. For the digital SOTA case, the complexity of solving symmetric, positive definite linear systems, matrix inverse, Lyapunov equation, and matrix determinant problems are respectively for algorithms based on: conjugate gradient method [49], fast matrix multiplication/inverse [47], Bartels-Stewart algorithm [5], and Cholesky decomposition [17]. $\omega \approx 2.3$ denotes the matrix multiplication constant.

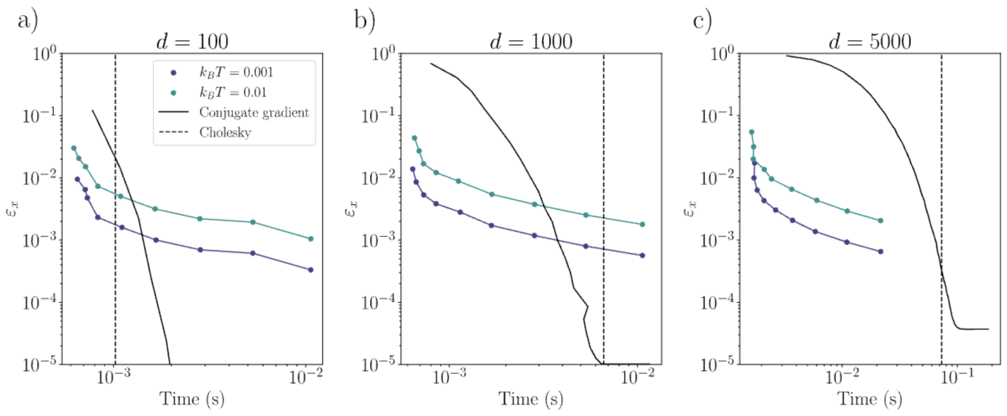

Figure 3: **Comparison of the error ε_x of the thermodynamic algorithm (TA) to solve linear systems with the conjugate gradient method and Cholesky decomposition as a function of total runtime.** The TA is shown for different values of k_BT (units of $1/\gamma$ where γ is the damping rate) for each dimension in $\{100, 1000, 5000\}$. Random matrices are drawn from the Wishart distribution and then mixed with the identity such that their condition numbers are respectively 120, 1189, 5995.

- We identified a class of algorithms called Thermodynamic AI algorithms that not only use stochasticity as a resource but also employ a Maxwell's demon subroutine to guide the random fluctuations in the right direction. These include generative diffusion models, Bayesian deep learning, and Monte Carlo inference.
- We also identified linear algebra primitives, like estimating matrix inverses and matrix determinants, as falling under the framework of Thermodynamic AI.
- All of these (seemingly distinct) algorithms can not only be run on a unified software platform, but can they can be run on a unified hardware platform.
- We also presented the first theoretical speedups for thermodynamic computing hardware, showing that certain linear algebra primitives could achieve a speedup on Thermodynamic AI hardware that grows linearly with matrix dimension.

In conclusion, various AI algorithms are simultaneously breaking practical barriers and yet also not reaching their full potential due to a mismatch with the underlying hardware. For example, approximation-free Bayesian deep learning is currently prohibitively slow on digital hardware [33]. Thermodynamic AI hardware, which is already seeing progress and interactive tools [30], appears to be a natural paradigm to unlock large speedups for these AI algorithms.

References

[1] N. A. Aadit, A. Grimaldi, M. Carpentieri, L. Theogarajan, J. M. Martinis, G. Finocchio, and K. Y. Camsari. Massively parallel probabilistic computing with sparse Ising machines. *Nat. Electron.*, 5(7):460–468, 2022.

[2] M. Aifer, K. Donatella, M. H. Gordon, T. Ahle, D. Simpson, G. E. Crooks, and P. J. Coles. Thermodynamic linear algebra. *arXiv preprint arXiv:2308.05660*, 2023.

[3] A. A. Alemi and I. Fischer. Therml: Thermodynamics of machine learning. *arXiv preprint arXiv:1807.04162*, 2018.

[4] Y. Bahri, J. Kadmon, J. Pennington, S. S. Schoenholz, J. Sohl-Dickstein, and S. Ganguli. Statistical mechanics of deep learning. *Annual Review of Condensed Matter Physics*, 11(1):501–528, 2020.

[5] R. H. Bartels and G. W. Stewart. Solution of the matrix equation $AX + XB = C$ [f4]. *Commun. ACM*, 15(9):820–826, 1972.

[6] A. Batou. An approximate Itô-SDE based simulated annealing algorithm for multivariate design optimization problems. *arXiv preprint arXiv:1901.10763*, 2019.

[7] M. Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.

[8] B. E. Boser, E. Sackinger, J. Bromley, Y. Le Cun, and L. D. Jackel. An analog neural network processor with programmable topology. *IEEE J. Solid-State Circuits*, 26(12):2017–2025, 1991.

[9] A. B. Boyd, J. P. Crutchfield, and M. Gu. Thermodynamic machine learning through maximum work production. *New J. Phys*, 24(8), 2022.

[10] K. Y. Camsari, B. M. Sutton, and S. Datta. P-bits for probabilistic spin logic. *Appl. Phys. Rev.*, 6(1), 2019.

[11] A. Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

[12] R. T. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

[13] T. Chen, E. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *International conference on machine learning*, pages 1683–1691. PMLR, 2014.

[14] C. D. Christ, A. E. Mark, and W. F. Van Gunsteren. Basic ingredients of free energy calculations: a review. *J. Comput. Chem.*, 31(8):1569–1582, 2010.

[15] P. J. Coles, C. Szczepanski, D. Melanson, K. Donatella, A. J. Martinez, and F. Sbahi. Thermodynamic AI and the fluctuation frontier. *arXiv preprint arXiv:2302.06584*, 2023.

[16] T. Conte, E. DeBenedictis, N. Ganesh, T. Hylton, J. P. Strachan, R. S. Williams, A. Alemi, L. Altenberg, G. E. Crooks, J. Crutchfield, et al. Thermodynamic computing. *arXiv preprint arXiv:1911.01968*, 2019.

[17] D. Dereniowski and M. Kubale. Cholesky factorization of matrices in parallel and ranking of graphs. In *Parallel Processing and Applied Mathematics: 5th International Conference, PPAM 2003, Czestochowa, Poland, September 7-10, 2003. Revised Papers 5*, pages 985–992. Springer, 2004.

[18] Y. Du, L. Du, X. Gu, J. Du, X. S. Wang, B. Hu, M. Jiang, X. Chen, S. S. Iyer, and M.-C. F. Chang. An analog neural network computing engine using CMOS-compatible charge-trap-transistor (CTT). *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(10):1811–1819, 2018.

[19] B. Dunham, D. Fridshal, R. Fridshal, and J. North. Design by natural selection. *Form and Strategy in Science: Studies Dedicated to Joseph Henry Woodger on the Occasion of his Seventieth Birthday*, pages 306–311, 1964.

[20] N. Ganesh. A thermodynamic treatment of intelligent systems. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–4, 2017.

[21] N. Ganesh. Thermodynamic intelligence, a heretical theory. In *2018 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–10, 2018.

[22] N. Ganesh. Rebooting neuromorphic design–a complexity engineering approach. In *2020 International Conference on Rebooting Computing (ICRC)*, pages 80–89. IEEE, 2020.

[23] E. Goan and C. Fookes. Bayesian neural networks: An introduction and survey. *Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018*, pages 45–87, 2020.

[24] S. Goldt and U. Seifert. Stochastic thermodynamics of learning. *Phys. Rev. Lett.*, 118:010601, 2017.

[25] G. Hinton. The forward-forward algorithm: Some preliminary investigations, 2022.

[26] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[27] M. D. Hoffman and A. Gelman. The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

[28] S. Hooker. The hardware lottery. *Commun. ACM*, 64(12):58–65, 2021.

[29] P. Horowitz and W. Hill. *The art of electronics; 3rd ed.* Cambridge University Press, 2015.

[30] M. Hunter Gordon, A. Tan, M. Aifer, K. Donatella, D. Melanson, G. Crooks, and P. J. Coles. Exploring thermodynamic AI. https://normalcomputing.substack.com/p/exploring-thermodynamic-ai.

[31] T. Hylton. Thermodynamic neural network. *Entropy*, 22(3):256, 2020.

[32] T. Hylton. Thermodynamic state machine network. *Entropy*, 24(6):744, 2022.

[33] P. Izmailov, S. Vikram, M. D. Hoffman, and A. G. G. Wilson. What are Bayesian neural network posteriors really like? In *International conference on machine learning*, pages 4629–4640. PMLR, 2021.

[34] C. Jarzynski. Nonequilibrium equality for free energy differences. *Physical Review Letters*, 78(14):2690, 1997.

[35] J. Kaiser, S. Datta, and B. Behin-Aein. Life is probabilistic-why should all our computers be deterministic? Computing with p-bits: Ising solvers and beyond. In *2022 International Electron Devices Meeting (IEDM)*, pages 21–4. IEEE, 2022.

[36] P. Kidger, J. Morrill, J. Foster, and T. Lyons. Neural controlled differential equations for irregular time series. *Advances in Neural Information Processing Systems*, 33:6696–6707, 2020.

[37] S. K. Kim, L. C. McAfee, P. L. McMahon, and K. Olukotun. A highly scalable restricted Boltzmann machine FPGA implementation. In *2009 International Conference on Field Programmable Logic and Applications*, pages 367–372. IEEE, 2009.

[38] S. Kirkpatrick, C. D. Gelatt Jr, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[39] D. P. Kroese, T. Brereton, T. Taimre, and Z. I. Botev. Why the Monte Carlo method is so important today. *Wiley Interdiscip. Rev.: Comput. Stat.*, 6(6):386–392, 2014.

[40] R. Kubo. The fluctuation-dissipation theorem. *Rep. Prog. Phys.*, 29(1):255, 1966.

[41] X. Li, T.-K. L. Wong, R. T. Chen, and D. K. Duvenaud. Scalable gradients and variational inference for stochastic differential equations. In *Symposium on Advances in Approximate Bayesian Inference*, pages 1–28. PMLR, 2020.

[42] V. K. Mansinghka et al. *Natively probabilistic computation*. PhD thesis, Citeseer, 2009.

[43] V. K. Mansinghka and E. M. Jonas. Combinational stochastic logic, Jan. 8 2013. US Patent 8,352,384.

[44] N. Metropolis. The beginning. *Los Alamos Science*, 15:125–130, 1987.

[45] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21(6):1087–1092, 1953.

[46] R. M. Neal. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.

[47] S. Robinson. Toward an optimal algorithm for matrix multiplication. *SIAM news*, 38(9):1–3, 2005.

[48] E. Säckinger, B. E. Boser, J. M. Bromley, Y. LeCun, and L. D. Jackel. Application of the ANNA neural network chip to high-speed character recognition. *IEEE Trans. Neural Netw.*, 3(3):498–505, 1992.

[49] J. R. Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

[50] V. N. Smelyanskiy, E. G. Rieffel, S. I. Knysh, C. P. Williams, M. W. Johnson, M. C. Thom, W. G. Macready, and K. L. Pudenz. A near-term quantum computing approach for hard computational problems in space exploration. *arXiv preprint arXiv:1204.2821*, 2012.

[51] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37, pages 2256–2265, 2015.

[52] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*, 2020.

[53] M. Verleysen and P. G. A. Jespers. An analog VLSI implementation of hopfield's neural network. *IEEE Micro*, 9(6):46–55, 1989.

[54] J. Voss. *An introduction to statistical computing: a simulation-based approach*. John Wiley & Sons, 2013.

[55] J. Weber. Fluctuation dissipation theorem. *Physical Review*, 101(6):1620, 1956.

[56] M. Welling and Y. W. Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[57] L. G. Wright, T. Onodera, M. M. Stein, T. Wang, D. T. Schachter, Z. Hu, and P. L. McMahon. Deep physical neural networks trained with backpropagation. *Nature*, 601(7894):549–555, 2022.

[58] W. Xu, R. T. Chen, X. Li, and D. Duvenaud. Infinitely deep bayesian neural networks with stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics*, pages 721–738. PMLR, 2022.

Supplementary Material

A Applications of Thermodynamic AI

In this Supplementary Material, we dive deeper into each of the following applications, describing how each of them falls under the framework of Thermodynamic AI:

1. Diffusion Models
2. Bayesian Deep Learning
3. Monte Carlo Inference
4. Annealing
5. Time Series Forecasting
6. Solving Linear Systems
7. Matrix Inversion
8. Solving Lyapunov Equations
9. Matrix Determinant

B Application: Diffusion Models

B.1 Background

Diffusion models [26, 52] are a state-of-the-art method for implementing generative models. Figure 4 shows the basic idea of diffusion models. These models add noise to data drawn from a data distribution p_{data} during a forward process that evolves from from $t = 0$ to $t = T$. A sample is then drawn from a noisy distribution, p_{noise}, and the process evolves in the reverse direction, from $t = T$ to $t = 0$, to generate a novel datapoint. Both the forward and reverse processes can be described by SDEs [52]. Namely, the reverse SDE is similar to the forward SDE except that it has an additional drift term. A typical forward and reverse SDE for diffusion models has the form:

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t \quad \text{(Forward)} \tag{8}$$

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t - g(t)^2\mathbf{s}_\theta(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\overline{\mathbf{w}}_t \quad \text{(Reverse)} \tag{9}$$

where $\mathbf{x}$ is the continuous state variable. In the reverse SDE, time t runs backwards and $\mathrm{d}\overline{\mathbf{w}}_t$ is a standard Brownian motion term when time runs backwards. The vector $\mathbf{s}_\theta(\mathbf{x}, t)$ is a model for the score function $\nabla_\mathbf{x} \log p_t(\mathbf{x})$ (the gradient of the logarithm of the probability distribution), and hence $\mathbf{s}_\theta(\mathbf{x}, t) \approx \nabla_\mathbf{x} \log p_t(\mathbf{x})$.

At a high level, the forward process provides training data to train a neural network, whose job is to output $\mathbf{s}_\theta(\mathbf{x}, t)$. This neural network is called the score network. Once the score network is trained, it can be used to guide the reverse process to generate novel samples.

In state-of-the-art diffusion models, the sampling rate still remains fairly slow. Hence, any means to speed up sampling rate could significantly improve this technology, and this is where Thermodynamic AI Hardware could help.

B.2 Fitting into our framework

Equations (8) and (9) each fall under the our framework, e.g., as special cases of Eq. (3). To make this more clear, one can do a change of variables $\tau = T - t$ in the reverse process and rewrite the diffusion model SDE equations as:

$$\mathrm{d}\mathbf{x} = f(t)\mathbf{x}\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}_t \quad \text{(Forward)} \tag{10}$$

$$\begin{aligned}\mathrm{d}\mathbf{x} = &-f(T-\tau)\mathbf{x}\mathrm{d}\tau + g(T-\tau)^2\mathbf{s}_\theta(\mathbf{x}, T-\tau)\mathrm{d}\tau \\ &+ g(T-\tau)\mathrm{d}\mathbf{w}_\tau \quad \text{(Reverse)}\end{aligned} \tag{11}$$

In this case, both t and τ run forward in time, i.e., dt and $d\tau$ are positive increments in these equations. Note that $\mathrm{d}\mathbf{w}_\tau$ appears in this equation since $\mathrm{d}\mathbf{w}_\tau = \mathrm{d}\overline{\mathbf{w}}_t$. Since the time variables

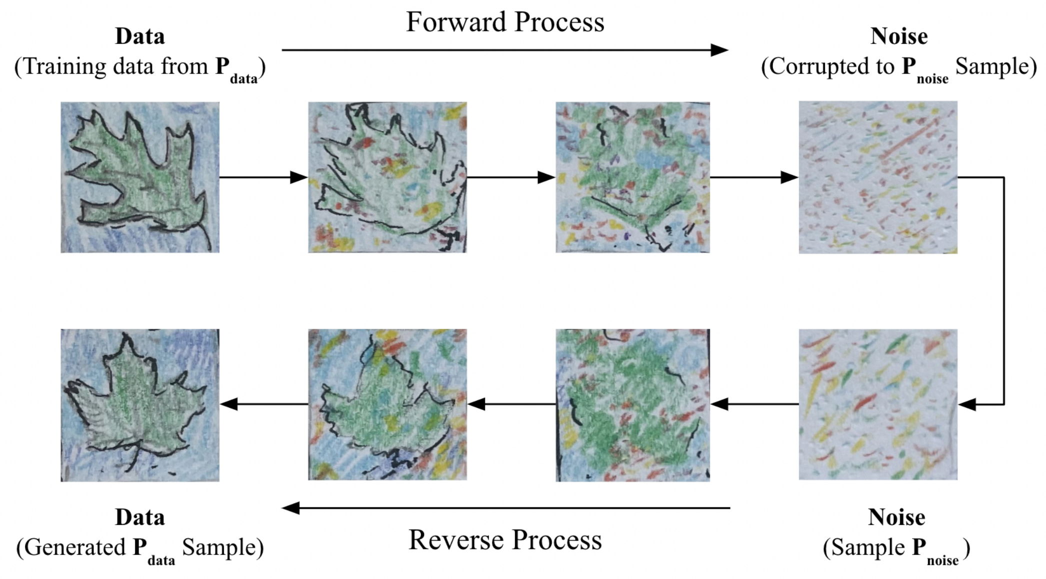
Data
(Training data from $\mathbf{P}_{\text{data}}$)
Forward Process
Noise
(Corrupted to $\mathbf{P}_{\text{noise}}$ Sample)
Data
(Generated $\mathbf{P}_{\text{data}}$ Sample)
Reverse Process
Noise
(Sample $\mathbf{P}_{\text{noise}}$)

Figure 4: Diffusion models add noise to data drawn from a distribution p_{data} during a forward process, providing data to train a score network. A sample is then drawn from a noisy distribution, p_{noise}, and the process is reversed using the trained score network to generate a novel datapoint.

in these equations run forward in time, then they can be directly implemented in physical analog hardware, since time always runs forward in the physical world.

The key to fitting diffusion models into our framework is to make to following mapping:

$$\text{(diffusion process)} \leftrightarrow \text{(s-mode device)} \tag{12}$$
$$\text{(score network)} \leftrightarrow \text{(Maxwell's demon device)} \tag{13}$$

The mathematical diffusion process in diffusion models can be mapped to the physical diffusion process in the s-mode device. Similarly, the score vector outputted by the score network corresponds to the vector $\mathbf{d}(t, \mathbf{v}(t))$ outputted by the MD device. Hence, diffusion models fit in our framework for Thermodynamic AI systems.

B.3 Description of Diffusion Hardware

Figure 5 gives a schematic diagram of a Thermodynamic Diffusion Model. A variety of different physical paradigms can be used to implement this hardware, such as analog electrical circuits or continuous-variable optical systems. Hence, we describe the system an abstract level.

As shown in Figure 5 the physical system has multiple degrees of freedom (DOFs) - which essentially correspond to the s-modes. The number of DOFs matches the dimensionality of the data, i.e., the number of features in the data. Each DOF has a continuous state variable, and that variable evolves according to a differential equation that, in general, could have both a diffusion and drift term. A function generator can multiply these diffusion and drift terms by arbitrary time-dependent functions ($h_j(t)$ and $k_j(t)$ respectively). The problem geometry, associated with a given dataset, can be uploaded onto the device by selectively connecting the various DOFs, which mathematically couples the differential equations of the various DOFs. After some encoding, a datapoint from the dataset of interest can be uploaded to the device by initializing the values of the continuous state variables to be the corresponding feature values of the datapoint. Similarly, data can be downloaded (and decoded) from the device by measuring the values of the continuous state variables after some time evolution.

In addition, the reverse process uses a trained score network. The inputs to the score network are the values of the continuous state variables at some time t, and the output is the value of the score. The jth component of the score, $s_j(t)$, gets added as a drift term in the evolution of the jth DOF.

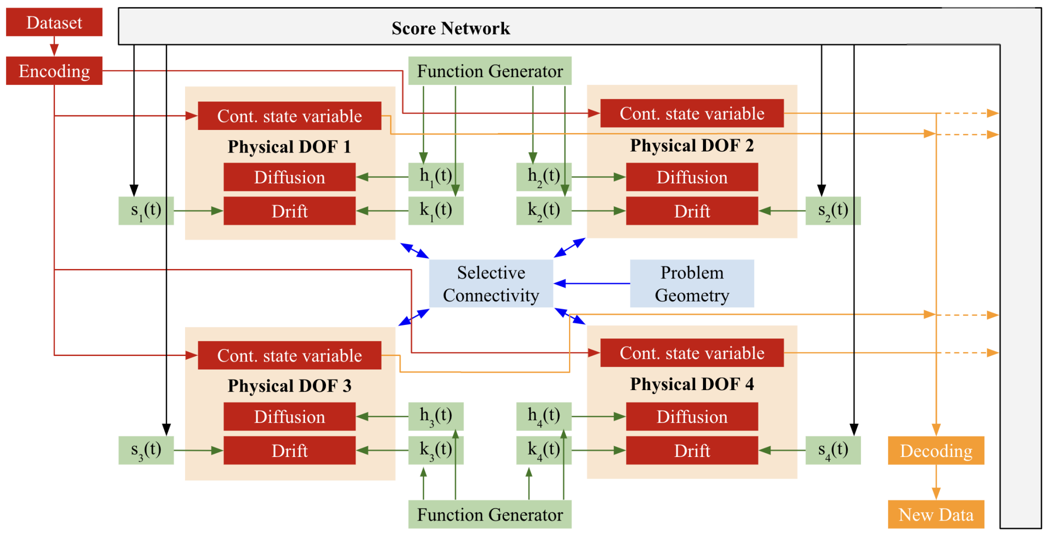

Figure 5: Schematic diagram of a Thermodynamic Diffusion Model. For simplicity, we show the case of four degrees of freedom (DOFs), where each DOF corresponds to an s-mode. These s-modes can be constructed as described in Sec. 3.

The score network acts as a Maxwell's Demon that continuously monitors the physical system and appropriately adapts the drift term to reduce the physical system's entropy.

B.4 Analog Score Network

We described in Sec. 4 how the Maxwell's Demon can take many physical forms, and the same holds for the score network. This includes the possibility of using a digital score network in conjunction with an analog s-mode system. However, this can lead to latency issues, whereby the communication between the score network and the s-mode system has some time delay. Hence, we highlight here the possibility of using an analog architecture for the score network, which could address the latency issue. Figure 6 shows an analog circuit for the score network. This circuit is based on decomposing the score prediction based on the total derivative formula [15].

C Application: Bayesian Deep Learning

C.1 Background

Machine learning systems such as neural networks are known to often be overconfident in their predictions. For low-stakes applications, this may not be a major issue. However, some applications, such as self-driving cars and medical diagnosis, are higher stakes in the sense that making a wrong decision can lead to consequences relevant to human life. Overconfidence can be catastrophic for these high-stakes applications.

At a technical level, this overconfidence often arises because neural networks are trained on limited amounts of training data. These training data points live in a vast feature space. Hence it is common for some regions of feature space may not be represented by the training data, i.e., these regions may be far away from the training data. When it comes time to test the trained neural network on testing data, the testing data could be in a region that is far away from training data, and yet the neural network will still attempt to make a prediction in this case. Because the neural network is not familiar with these regions, the neural network is not aware that it should be careful when making predictions for them.

One strategy for dealing with overconfidence is uncertainty quantification (UQ). UQ aims to quantify the uncertainty of the predictions made by the neural network. UQ is useful for high-stakes applications (e.g., cancer detection in medicine) because it provides guidance for when the user

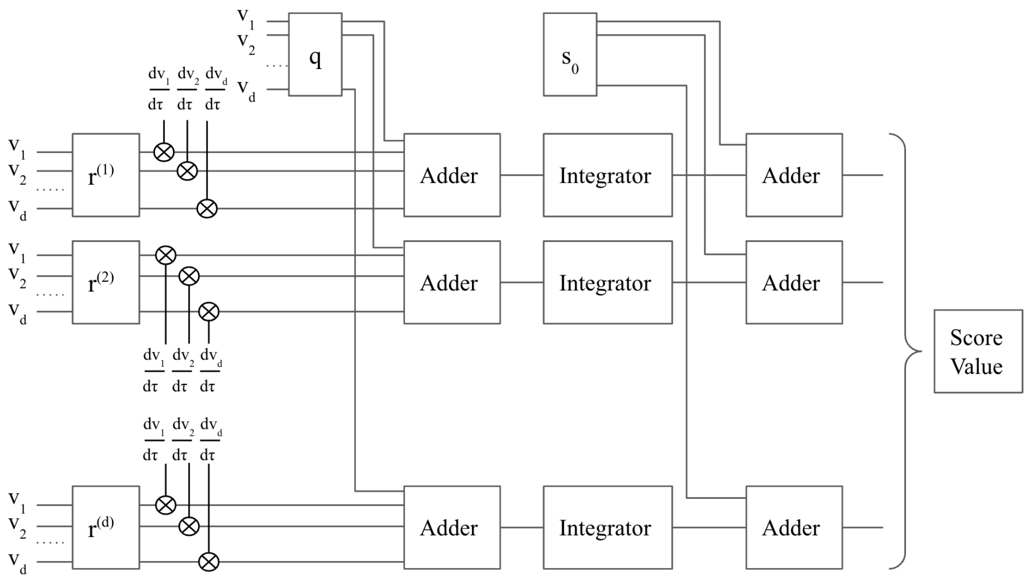

Figure 6: Schematic circuit diagram for an analog score network based on the total derivative approach. This diagram represents the process used to obtain score values during the evolution of the reverse diffusion process.

should defer to human judgement over the machine's predictions. UQ is widely recognized as making machine learning more reliable and trustworthy.

Several different methods exist for UQ in machine learning. A simple example of UQ is adding confidence intervals to the predictions made by the neural network.

A more sophisticated and rigorous approach to UQ is the Bayesian framework. The Bayesian framework quantifies uncertainty by accounting for prior knowledge (often called the prior distribution) and updates that knowledge due to data or observations (often called the posterior distribution). Bayesian methods aim to quantitatively capture knowledge in the form of probability distributions.

C.2 Neural Differential Equations

A continuous-time approach to Bayesian deep learning was recently developed [58], and hence we consider that approach in what follows.

Neural Ordinary Differential Equations (Neural ODEs) [12] are a continuous depth version of neural networks. The values of the hidden units are denoted h_t and the values of the weights are denoted w_t. In general, both of these quantities depend on time, and evolve according to the coupled differential equations:

$$\frac{d}{dt}\begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ f_w(t, w_t) \end{bmatrix}$$

Hence, a forward pass through the neural network involves integrating this system of differential equations.

Neural Stochastic Differential Equations (Neural SDEs) [58] are a continuous depth version of Bayesian neural networks. Once again, the system evolves according to a system of coupled differential equations. However, the weights of the neural ODE evolve in a stochastic manner:

$$d\begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ f_w(t, w_t) \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ g_w(t, w_t) \end{bmatrix} dB \tag{14}$$

where dB is a Brownian motion term. Equation (14) provides the basis for Bayesian deep learning in continuous time.

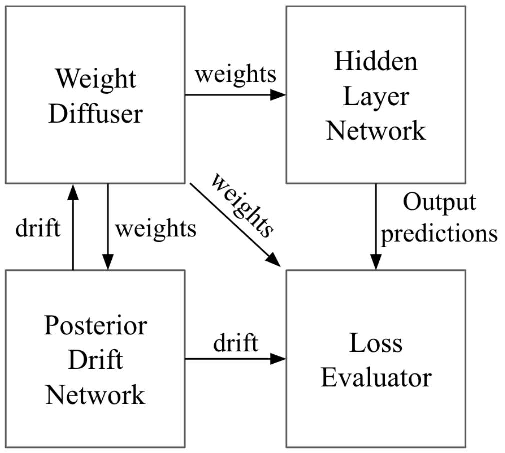

Figure 7: Overview of Thermodynamic Bayesian deep learning, showing how the four subroutines interact and feed signals to each other.

For the posterior distribution we can specialize Eq. (14) to the following form:

$$d\begin{bmatrix} h_t \\ w_t \end{bmatrix} = \begin{bmatrix} f_h(t, h_t, w_t) \\ \mathrm{NN}_\phi(t, w_t, \phi) + w_t \end{bmatrix} dt + \begin{bmatrix} \mathbf{0} \\ \sigma I_d \end{bmatrix} dB \tag{15}$$

The posterior distribution needs to be highly expressive. Hence, the SDE must be more complicated than that of the prior distribution. The drift term for the weights is therefore described by a neural network NN_ϕ with trainable parameters ϕ. Note that the trainable parameters in this model include both the parameters ϕ appearing in the drift term as well as the initial condition w_0 on the weights. The neural network NN_ϕ can be referred to as the Posterior Drift Network (PDN), since it determines the drift associated with the posterior distribution.

C.3 Subroutines in Bayesian deep learning hardware

The Thermodynamic AI system for Bayesian deep learning consists of four subroutines. Each of these subroutines can correspond to a physical analog device, or some subset of the subroutines may be stored in and processed on a digital device. The four subroutines include the following:

1. Hidden layer network
2. Weight diffuser
3. Posterior drift network
4. Loss evaluator

Figure 7 illustrates how these four subroutines interact with each other. The Weight Diffuser (WD), which can represent both the prior distribution and the posterior distribution, feeds weight values to the Hidden Layer Network (HLN). The WD also communicates back-and-forth with the Posterior Drift Network (PDN), in which the WD feeds weight values to the PDN and the PDN feeds drift values to the WD. The Loss Evaluator (LE) takes in signals from all three of the other subroutines - the HLN, the WD, and the PDN - in order to evaluate the loss function.

C.4 Fitting into our Thermodynamic AI framework

We make the following mapping in order to fit this into our framework:

$$\text{(weight diffuser)} \leftrightarrow \text{(s-mode device)} \tag{16}$$

$$\text{(posterior drift network)} \leftrightarrow \text{(Maxwell's demon device)} \tag{17}$$

The weight diffuser corresponds to the s-mode device, and the posterior drift network corresponds to the Maxwell's demon device.

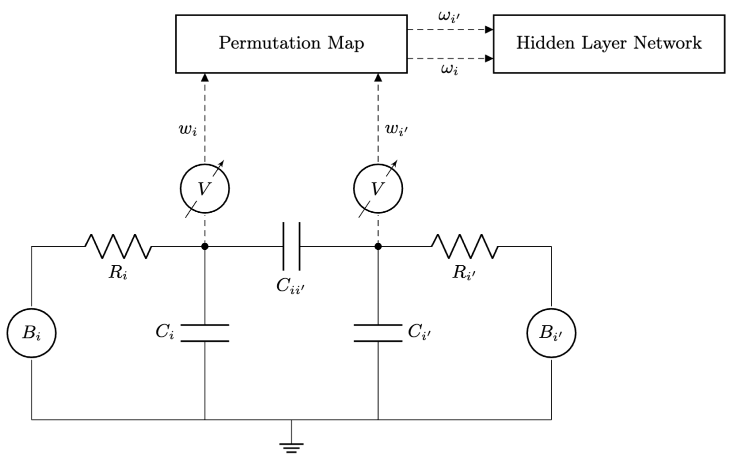

Figure 8: Outputting weights from the weight diffuser device to the HLN device. Shown here is the simplified case for two unit cells of the weight diffuser, although the concept applies to an arbitrary number of unit cells. A permutation map (which physically correspond to a wire routing scheme) is shown to provide flexibility for how the weights $\mathbf{w}$ outputted by the WD get incorporated as weights ω in the HLN.

The weight diffuser uses s-mode dynamics to sample weight trajectories w_t, which are then imported into the HLN. The Maxwell's demon is used to produce complex s-mode dynamics that produce weight trajectories according to a posterior distribution.

C.5 Description of Bayesian Deep Learning Hardware

Figure 8 illustrates a possible hardware implementation of the weight diffuser. As shown, the outputs of the weight diffuser are supplied to the Hidden Layer Network device.

The weight diffuser can be an analog circuit, with variables $v(t)$, equal the voltage values across a set of capacitors located in a series of unit cells. These voltage values diffuse according to a stochastic process inside the circuit with analog noise sources B_i, and are output to a hidden layer network as time-continuous weight trajectories. The same diffusing cell can generate prior and posterior samples, depending on whether the posterior drift network applies a demon voltage vector to the circuit.

The hidden layer network may also be an analog circuit, whose dynamics can be modeled as a neural ODE dependent on the sampled weight trajectory outputted by the weight diffuser.

D Application: Monte Carlo Inference

D.1 Background

Monte Carlo algorithms [44] have become widely used, finding application in finance, physics, chemistry, and more recently, machine learning [39]. Monte Carlo algorithms provide a simple procedure for approximating integrals involving probability distributions. Suppose we have a probability distribution $\pi(\mathbf{x})$ with $\mathbf{x} \in \mathcal{X}$ the sample space, and suppose we want to find the expectation value of some function $f(\mathbf{x})$ with respect to π. The Monte Carlo method consists in approximating the integral with a simple sample average,

$$\int f(\mathbf{x})\pi(\mathbf{x})d\mathbf{x} \approx \frac{1}{M}\sum_{i=1}^{M} f(\mathbf{x}_i), \tag{18}$$

where the samples $\mathbf{x}_i$ are distributed according to π. The computational bottleneck has been transformed from integration to sampling. The best methods for sampling from π will depend on the form of π.

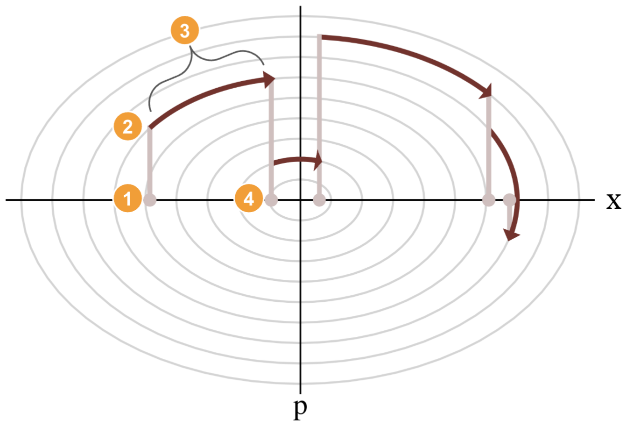

Figure 9: Phase space picture of the HMC algorithm, for a 1D parabolic potential. Step 1: Initialize at the most recent chain position x_i. Step 2: Randomly sample momentum p from a known probability distribution. Step 3: Integrate Hamilton's equations. Step 4: accept or reject the move from x_i to x'. Figure adapted from [7].

Markov Chain Monte Carlo (MCMC) is one popular strategy for constructing samplers [45], and can be applied whether the state space $\mathcal{X}$ is discrete or continuous. This strategy involves setting up a chain of dependent samples, such that over a long enough time, the set of samples becomes distributed according to $\pi(\mathbf{x})$. In other words, it operates by constructing a Markov chain that has the target distribution π as its stationary distribution. An MCMC algorithm is defined by making a choice of conditional distribution $g(\mathbf{x}'|\mathbf{x})$ and a choice of initial state $\mathbf{x}_0$, then repeating the following for each iteration i [54]:

1. Initialize at position $\mathbf{x}_i$
2. Draw a sample from g as $\mathbf{x}' \sim g(\mathbf{x}'|\mathbf{x}_i)$
3. Compute the probability Π_a of accepting $\mathbf{x}'$ as the next state in the chain. The probability is given by
$$\Pi_a(\mathbf{x}', \mathbf{x}_i) = \min\left(1, \frac{\pi(\mathbf{x}')g(\mathbf{x}_i|\mathbf{x}')}{\pi(\mathbf{x}_i)g(\mathbf{x}'|\mathbf{x}_i)}\right) \tag{19}$$
4. Draw a random number r uniformly from the unit interval $[0, 1]$. If $r < \Pi_a$, accept the move by setting $\mathbf{x}_{i+1} = \mathbf{x}'$. Otherwise, stay at the current position by setting $\mathbf{x}_{i+1} = \mathbf{x}_i$.

Note that one should choose the conditional distribution g such that it is easier to sample from than the target distribution π.

In the following sections we will present how the framework of thermodynamic AI systems naturally encompasses Monte Carlo algorithms, focusing on two key algorithms: Langevin Monte Carlo (LMC) and Hamiltonian Monte Carlo (HMC) [7].

D.1.1 Hamiltonian Monte Carlo

HMC has gradually become one of the most widely used MCMC algorithms for statistical analysis and learning thanks to its computational efficiency and sample quality [46]. The markov chain in HMC is constructed by proposing new samples using a combination of gradient information and Hamiltonian dynamics. The key idea behind HMC is to introduce fictitious momentum variables $\mathbf{p}$, to which $\mathbf{x}$ is coupled according to Hamilton's equations:

$$\frac{\partial \mathbf{x}}{\partial t} = \frac{\partial}{\partial \mathbf{p}} H(\mathbf{x}, \mathbf{p}) \tag{20}$$
$$\frac{\partial \mathbf{p}}{\partial t} = -\frac{\partial}{\partial \mathbf{x}} H(\mathbf{x}, \mathbf{p}) \tag{21}$$

with the Hamiltonian H defined as $H(\mathbf{x}, \mathbf{p}) = -\log \pi(\mathbf{x}) + \mathbf{p}^T M \mathbf{p}/2$, with a potential term $U(\mathbf{x}) = -\log \pi(\mathbf{x})$ corresponding to the landscape of the target probability distribution in log space, and

M a mass matrix. At each iteration of the HMC algorithm, a new sample is proposed by integrating the equations of motion in phase space over a fixed time interval τ. As such, HMC is referred to as a gradient-augmented MCMC method, where information on the gradient of the log-probability is integrated in the chain. The proposed sample is then accepted or rejected using the Metropolis-Hastings acceptance criterion, which in the case of Hamiltonian dynamics reduces to

$$\Pi_a(\mathbf{x}', \mathbf{x}) = \frac{\pi(\mathbf{x}')}{\pi(\mathbf{x})} \tag{22}$$

since the dynamics are reversible which gives $g(\mathbf{x}', \mathbf{x}) = g(\mathbf{x}, \mathbf{x}')$. The gradient information in HMC is used to define the direction of the proposed updates, allowing the Markov chain to efficiently explore regions of high probability mass. Therefore, regions of low probability mass may be avoided, thus allowing the Markov chain to escape from local modes and explore the target distribution more effectively unlike random walk MCMC methods. One may already see the connection with force-based MD, introduced in section IX, that is made more explicitly in the next subsection.

A more elaborate version of HMC has been developed and is also widely used in statistics, coined the No U-Turn sampler (NUTS). NUTS has the advantage of automatically tuning the step size and the trajectory during the sampling process, making it easier to use in practice [27].

D.1.2 Stochastic Gradient Hamiltonian Monte Carlo

Stochastic Gradient Hamiltonian Monte Carlo (SGHMC) [13] is an extension of HMC, proposed to use HMC efficiently on large problem sizes where computing exactly the gradient of the log probability $\nabla_{\mathbf{x}} \log \pi(\mathbf{x}) = \nabla U(\mathbf{x})$ (which is necessary to compute the dynamics) cannot be performed. Indeed, this gradient can be expressed as

$$\nabla U(\mathbf{x}) = -\sum_{\mathbf{x}_i \in \mathcal{D}} \nabla \log \pi(\mathbf{x}_i|\mathbf{x}) - \nabla \log p(\mathbf{x}). \tag{23}$$

for points $\mathbf{x}_i \in \mathcal{D}$, with $\mathcal{D}$ the set of observations. For large problem sizes, this quickly becomes intractable. To overcome this, the gradient may be approximated by uniformly sampling points $\mathbf{x}_i \in \tilde{\mathcal{D}}, \tilde{\mathcal{D}} \subset \mathcal{D}$:

$$\nabla \tilde{U}(\mathbf{x}) = -\frac{|\mathcal{D}|}{|\tilde{\mathcal{D}}|} \sum_{\mathbf{x}_i \in \tilde{\mathcal{D}}} \nabla \log \pi(\mathbf{x}_i|\mathbf{x}) - \nabla \log p(\mathbf{x}). \tag{24}$$

Assuming the x_i are independent, the central limit theorem leads to:

$$\nabla \tilde{U}(\mathbf{x}) \approx \nabla U(\mathbf{x}) + \mathcal{N}(0, V(\theta)) \tag{25}$$

with V the covariance of Gaussian noise with zero mean coming from the stochastic gradient approximation. In Ref. [13], it was shown that one can add a friction term to counterbalance the effect of the noise coming from the stochastic gradient, thus obtaining the dynamical equations for SGHMC with friction:

$$\mathrm{d}\mathbf{x} = M^{-1}\mathbf{p}\,\mathrm{d}t \tag{26}$$

$$\mathrm{d}\mathbf{p} = -[\nabla U(\mathbf{x}) + BM^{-1}\mathbf{p}]\,\mathrm{d}t + \sqrt{2}B\mathrm{d}\mathbf{w} \tag{27}$$

where M is a mass matrix, $B = V(\theta)/2$ is the diffusion matrix, $\nabla U(\mathbf{x}) = \mathbf{f}(\mathbf{x})$ is the force and $\mathbf{w}$ is the Wiener process. The stationary distribution for $\mathbf{x}$ obtained at long times corresponds to the target probability distribution $\pi(\mathbf{x}) = \exp\{-U(\mathbf{x})\}$. In fact, looking at Eq. (27), one can see the close connection with Eqs. (9) and (15).

D.2 Connection to Langevin Monte Carlo

There is a close connection between SG-HMC and Langevin Monte Carlo (LMC), in particular with a variation of LMC known as stochastic gradient Langevin dynamics (SGLD). SGLD is a procedure for Bayesian posterior sampling of the parameters of a machine learning model. As in [56], let θ be the parameters of the model, let $p(\theta)$ be a prior distribution on the parameters, and let $p(x|\theta)$ be the probability of data point x given that our model is parameterized by θ. Similarly to SGHMC, we can imagine introducing the N data points randomly in small batches of size n. Then the SGLD dynamics are specified by the update equation

$$\Delta\theta_t = \frac{\epsilon_t}{2}\left(\nabla \log p(\theta_t) + \frac{N}{n}\nabla \log p(x_{ti}|\theta_t)\right) + \mathcal{N}(0, \epsilon_t), \tag{28}$$

where ϵ_t is a time-dependent step size. The noise term prevents the the parameters from freezing at a particular value, instead being spread according to the posterior distribution. From this perspective we can see thermodynamic fluctuations as a resource for posterior inference.

We can further cast this equation in terms of the force framework introduced in section 6. Since we can treat the logarithm of a distribution as an energy, we can write $U(t, \theta) = \log p(\theta) + \frac{N}{n}\log p(x_{ti}|\theta_t)$, so that $U(t, \theta)$ is a time dependent energy function. Then we have

$$\mathrm{d}\theta = \frac{\epsilon_t}{2}\nabla U(t, \theta)\mathrm{d}t + \sqrt{\epsilon_t}dW. \tag{29}$$

Thus the data becomes part of a time dependent diffusion vector.

D.3 Fitting into our Thermodynamic AI Framework

Figure 10 shows how these algorithms can fit into our framework. We propose the following mapping to the s-unit formalism:

$$\text{(momentum device)} \leftrightarrow \text{(s-mode device, with a noise source and an injected value for the force)} \tag{30}$$

$$\text{(position device)} \leftrightarrow \text{(latent variable stored in the Maxwell's Demon memory)} \tag{31}$$

This mapping refers to how the differential equations in, for example, Eq. (26) and Eq. (27) are mapped to the devices in the Thermodynamic AI system. Assuming the noise is uncorrelated, one can set B in Eq. (27) to a scalar value. M will therefore be given by the connectivity of the momentum device.

D.4 Description of Monte Carlo Hardware

The coupled differential equations described above are amenable to being implemented on Thermodynamic AI hardware. Implementing the SGHMC algorithm on digital hardware requires one to compute the derivatives of position and momentum, which involve diagonalizing matrices, hence having a computation cost in $O(n^3)$ in the general case, with n the number of data points.

By implementing SGHMC in Thermodynamic AI hardware, this scaling may be eased. This can help in alleviating the bottleneck that is sampling for many applications. We coin Thermodynamic Monte Carlo (TMC) for the implementation of Monte Carlo algorithms in Thermodynamic AI hardware.

For the hardware implementation of SGMHC, one can consider two devices: one whose state variable is $\mathbf{x}$, and one whose state variable is $\mathbf{p}$. Ultimately, one is interested in obtaining values of $\mathbf{x}$, whose evolution is dictated by the dynamics of $\mathbf{p}$. To unify the SGHMC and SGLD approaches, we propose a single hardware paradigm.

This platform is depicted in Fig. 10. The force is calculated by the MD, which is then fed into the momentum device. The momentum vector is fed in real time to the integrator, the result of which is the position vector which is fed back into the Maxwell's demon for storage.

In the case of SGHMC, a noisy estimate of the force is calculated, which is then fed into the momentum device, which has to have no Brownian noise so that the correct SDE is implemented. The friction term may be implemented by dissipative elements, such as resistors.

In the case of SGLD, the force is calculated exactly, and then fed into the momentum device that will have both friction (that is high in this case) and Gaussian noise.

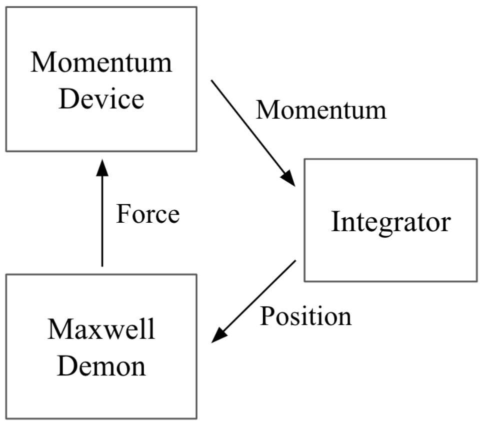

Figure 10: Schematic diagram for a Thermodynamic Monte Carlo device. The momentum device corresponds to an s-mode device, with s-modes constructed as described in Sec. 3. The Maxwell's demon is constructed via a force-based approach.

E Application: Annealing

E.1 Background

Many important problems can be phrased as optimization problems. Posing a problem as an optimization problem means defining a loss function on the space of potential solutions, such that the better the answer, the lower the corresponding value of the loss function. While many methods exist for solving optimization problems, a key difficulty that must be overcome is the existence of local minima in the loss function. These local minima show up whether the solution space is discrete or continuous. We outline a naive way to approach each domain, and how local minima thwart successfully finding the global optimum:

- For continuous problems, we might apply gradient descent [11]. Starting from a random location in solution space, the algorithm repeatedly computes the local gradient and uses it to travel down hill. If the algorithm reaches a local minimum, the gradient becomes zero, so the algorithm ceases to move.
- For discrete problems, we might apply local search [19]. Starting from a random configuration, we compute the value of all neighboring configurations; if any neighbor has a lower loss function value, the algorithm moves to that configuration. In a local minimum, no neighbor has a lower value of the loss, halting progress.

In both of these cases, we need something more to prevent becoming trapped in a local minimum.

The key missing ingredient is the ability to temporarily move to a solution or configuration that has a **larger** value of the loss. After such a temporary up-hill traversal, the algorithm has a chance to move into a neighboring, deeper minimum. This is where we can leverage thermal fluctuations as a resource for computation. If one perturbs the gradient direction (in continuous problems) or the loss values of the neighbors (for discrete problems), then the transition step no longer gets stuck in local minima. An illustration of this thermal escape from a minimum is illustrated in Figure 11.

On digital systems, one algorithm which takes advantage of thermal fluctuations is Simulated Annealing, initially proposed in 1983 [38]. Annealing is a process in which a metal is slowly cooled from a high temperature to increase its strength. Simulated Annealing makes an analogy between such annealing of a metal and optimization: the bonds in the metal are analogous to the loss function, while the strengthening of the metal is analogous to finding a better optimum.

In a physical system, all these perturbations can be provided by thermal fluctuations. In the next section we formalize the connection between the stochastic dynamics of a physical system and the Simulated Annealing algorithm.

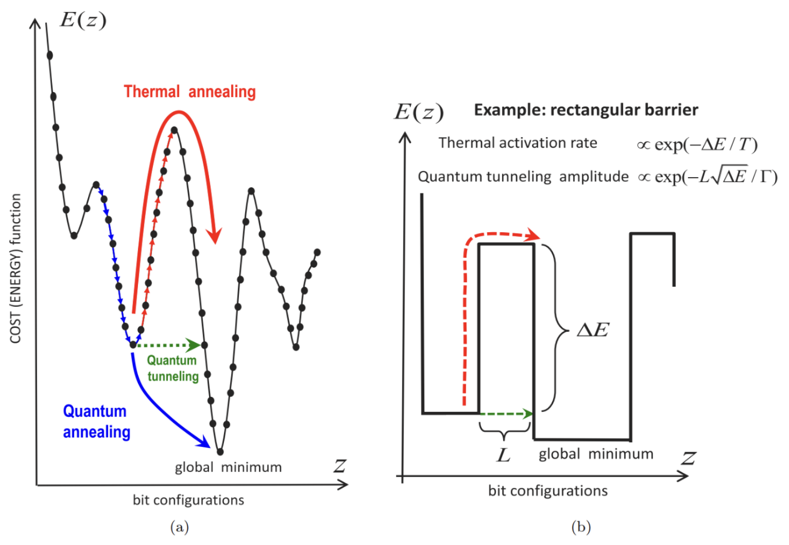

Figure 11: Illustration of thermal and quantum fluctuations enabling escape from local minima. (a) Thermal fluctuations provide temporary energy boosts that enable climbing over barriers. Quantum fluctuations make it possible to tunnel through barriers. (b) For a simple rectangular barrier, we can write down the transition probabilities provided by both thermal and quantum fluctuations. We see that they scale differently: thermal fluctuations do not depend on the width of the barrier, while quantum tunneling does. Figure from Ref. [50].

E.2 SDE approach to simulated annealing

Reference [6] provides a mathematical framework for simulated annealing based on SDEs. Let us discuss that framework now.

Suppose that we have an optimization problem, where the loss function $\mathcal{L}(\mathbf{x})$ of interest is a continuous function of an N-dimensional state variable $\mathbf{x}$. Here, $\mathbf{x}$ is the variable that one is optimizing over to solve the optimization problem in the context of simulated annealing. In this setting, one can propose a coupled system of equations for the state variable $\mathbf{x}(t)$ and an auxiliary variable $\mathbf{p}(t)$:

$$d\mathbf{x}(t) = \mathbf{p}(t)dt \tag{32}$$

$$d\mathbf{p}(t) = -\nabla\mathcal{L}(\mathbf{x})dt - \frac{1}{2}D\,\mathbf{p}(t)dt + S\,d\mathbf{W} \tag{33}$$

Here, $\mathbf{W}$ is an N-dimensional Brownian motion, S an $N \times N$ dimensional lower-triangular matrix, and $D = SS^{\intercal}$. We call the first differential equation the optimization ODE, while we call Eq. (33) the auxiliary SDE.

The dynamics on the state variable $\mathbf{x}(t)$ are effectively stochastic, as it is coupled to the auxiliary variable evolving via the auxiliary SDE.

In the long-time limit, the state variable $\mathbf{x}(t)$ is distributed according to a Boltzmann probability distribution constructed from the loss function $\mathcal{L}$:

$$\mathbf{x}(t \to \infty) \sim \exp(-\mathcal{L}(\mathbf{x})) \tag{34}$$

As such, the long-run samples will most often be concentrated around the extrema of the loss function $\mathcal{L}$, allowing one to identify the minima or maxima. As the state of $\mathbf{x}(t)$ is probabilistic, the entire extrema landscape of $\mathcal{L}$ can be explored.

E.3 Fitting into our Thermodynamic AI framework

Equations (32) and (33) fit into our framework for Thermodynamic AI hardware. Specifically, we have the following mapping to our hardware:

$$\text{(auxiliary SDE)} \leftrightarrow \text{(s-mode device)} \tag{35}$$

$$\text{(optimization ODE)} \leftrightarrow \text{(latent variable evolution in Maxwell's demon device)} \tag{36}$$

The idea is that the auxiliary SDE describing the evolution of $\mathbf{p}$ can be performed on the s-mode device. Here, S would correspond to the coefficient $C(t)$ in our hardware, and $-(1/2)D$ would correspond to the coefficient $A(t)$ in our hardware.

In addition, $-\nabla\mathcal{L}(\mathbf{x})$ would correspond to the demon vector $\mathbf{d}$ in our hardware. The optimization ODE then maps onto the evolution of the latent variable in the Maxwell's demon device. Note that this employs the framework discussed in Sections 4 and 6 involving a forced-based Maxwell's demon. Also, note that the mass matrix that appears in our framework is set to be the identity for this application: $M = I$.

F Application: Time Series Forecasting

F.1 Background

As a final application, we consider analysis of time-series data. Time-series data provide an important application relevant to financial analysis, market prediction, epidemiology, and medical data analysis. In many case one has data at particular time points that may be at irregular time intervals, and one wishes to have a model that makes a predictions at all times and hence one that interpolates between the datapoints. In addition, one may want to a model that extrapolates beyond the data, e.g., to make predictions about the future where no data is available.

Discrete neural networks, such as recurrent neural networks, have been used in the past for interpolating and extrapolating time-series data. However, latent ordinary differential equations (latent ODEs) [12] have been shown to outperform recurrent neural networks at this task. One can view a latent ODE as a parameterized ODE, where the parameters are trained in order to fit the time-series data (according to some loss function). More recently, latent SDEs have been explored for fitting and extrapolating time-series data [41].

F.2 Fitting into our Thermodynamic AI framework

In what follows, we discuss using Thermodynamic AI hardware as either a latent ODE or latent SDE, in order to interpolate and extrapolate a time-series dataset.

For concreteness, consider the case of a latent SDE. In this case, the idea is that the SDE should have trainable parameters that allow it to be fit to the data. This fits well with our Thermodynamic AI hardware, since one can use an s-mode device combined with a (parameterized) Maxwell's demon device to generate a parameterized SDE. For example, the overall dynamics associated with this parameterized SDE could be given by the following equation, which is a special case of Eq. (3):

$$d\mathbf{v}(t) = (A(t)\mathbf{v}(t) + \mathbf{b}(t) + D(t)\mathbf{d}_\theta(t, \mathbf{v}(t)))dt + C(t)d\mathbf{w}\,. \tag{37}$$

Figure 12 provides a schematic diagram for a potential approach. The overall model in Fig. 12 has three subroutines:

1. Encoder
2. Latent Thermodynamic AI hardware
3. Decoder

The training data are provided as observations from some time series. These time-series observations are fed into an encoder. The encoder has free parameters that can be trained. For example, the encoder could be a recurrent neural network. The output of the encoder can be the initial vector $\mathbf{h}(0)$ of the hidden layer values, or the output could be a probability distribution from which $\mathbf{h}(0)$ is sampled. If the encoder is stored on a digital device, its output can converted to an analog signal.

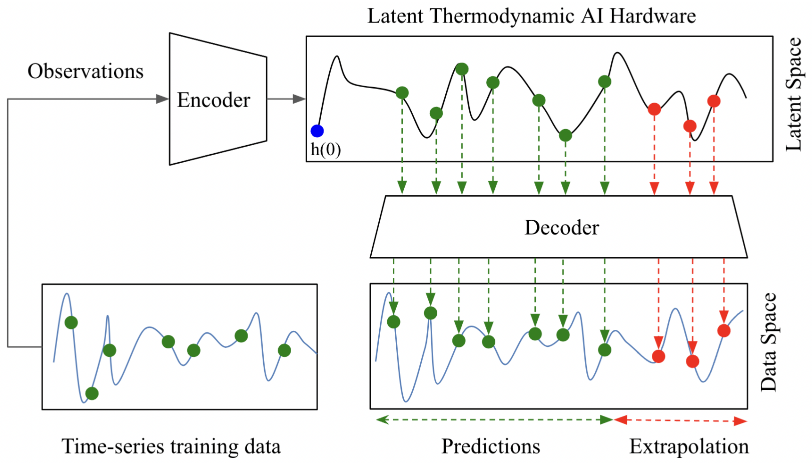

Figure 12: Illustration of latent Thermodynamic AI hardware for fitting and extrapolating time-series data.

The Thermodynamic AI hardware acts as the latent space for the latent ODE or latent SDE. This latent space is initialized to $\mathbf{h}(0)$ by the encoder. Then then hidden layer values evolve over time according to ODE or SDE that describes the system, such as the SDE in Eq. (37).

The hidden layer values $h(t_k)$ can be read off at a set $\{t_k\}$ of various times, e.g., by measuring the state variables of the s-mode system in the Thermodynamic AI hardware. This set $\{h(t_k)\}$ of values can be fed to a decoder. The decoder can have free parameters that will be trained. The outputs of the decoder correspond to predictions that the latent ODE model makes for the true time series. These predictions can go beyond the time interval associated with the observations, in which case the predictions correspond to extrapolated values.

A training process occurs where the parameters of the encoder, of the decoder, and of the Maxwell's demon in the Thermodynamic AI hardware optimized in order to minimize or maximize a loss function. This essentially corresponds to fitting the time-series data. Gradient based approaches such as the adjoint sensitivity method can be employed here.

G Application: Solving Linear Systems

We gave an overview of the Thermodynamic Linear Systems protocol in the main text. Here we provide a more detailed version of the protocol, as follows.

Linear System Protocol

1. Given a linear system $Ax = b$, set the potential of the device to

$$V(x) = \frac{1}{2}x^\intercal A x - b^\intercal x \tag{38}$$

at time $t = 0$.

2. Choose equilibration tolerance parameters $\varepsilon_{\mu 0}, \varepsilon_{\Sigma 0} \in \mathbb{R}^+$, and choose the equilibration time

$$t_0 \geq \widehat{t}_0, \tag{39}$$

where $\widehat{t}_0$ is computed from the system's physical properties or using heuristic methods based on Eqs. (43), (45). Allow the system to evolve under its dynamics until $t = t_0$, which ensures that $\left\|\langle x\rangle - A^{-1}b\right\| / \|A^{-1}b\| \leq \varepsilon_{\mu 0}$ and $\left\|\Sigma - \beta^{-1}A^{-1}\right\| / \|\beta^{-1}A^{-1}\| \leq \varepsilon_{\Sigma 0}$.

3. Choose error tolerance parameter ε_x and success probability P_ε, and choose the integration time
$$\tau \geq \widehat{\tau}, \tag{40}$$
where $\widehat{\tau}$ is computed from the system's physical properties, Eq. (43) or (45). Use an analog integrator to measure the the time average
$$\bar{x} = \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dt\, x(t), \tag{41}$$
which satisfies $\|A\bar{x} - b\| / \|b\| \leq \varepsilon_x$ with probability at least P_δ.

In order to implement the protocol above, the necessary values of $\widehat{t_0}$ and $\widehat{\tau}$ must be identified, which requires a more quantitative description of equilibration and ergodicity. To obtain such a description, a model of the system's microscopic dynamics may be introduced. Given that the system under consideration is composed of harmonic oscillators in contact with a heat bath, it is natural to allow for damping (i.e., energy loss to the bath) and stochastic thermal noise, which always accompanies damping due to the fluctuation-dissipation theorem [40, 55]. The Langevin equation accounts for these effects, and specifically we consider two common formulations, the overdamped Langevin (ODL) equation and the underdamped Langevin (UDL) equations.

The ODL equation for this system is
$$dx = -\frac{1}{\gamma}(Ax - b)dt + \mathcal{N}\left[0, 2\gamma^{-1}\beta^{-1}\, dt\right], \tag{42}$$
where $\gamma > 0$ is called the damping constant and $\beta = 1/k_B T$ is the inverse temperature of the environment. The system has a physical timescale (which is clear from dimensional analysis) that we call the *relaxation time* $\tau_\mathrm{r} = \gamma/\|A\|$. The condition number of A is $\kappa = \alpha_\mathrm{max}/\alpha_\mathrm{min}$, where $\alpha_1 \dots \alpha_d$ are the eigenvalues of A. With these definitions, we arrive at the following formulas for $\widehat{t_0}$ and $\widehat{\tau}$ in the overdamped case, which can be used in the above protocol:
$$\widehat{t_0} = \max\left\{\kappa\tau_\mathrm{r}\ln\left(\kappa\varepsilon_{\mu0}^{-1}\right), \frac{1}{2}\kappa\tau_\mathrm{r}\ln\left(2\kappa\varepsilon_{\Sigma0}^{-1}\right)\right\}, \quad \widehat{\tau} = \frac{2\kappa^2 d\,\|A\|}{\beta\|b\|^2\varepsilon_x^2(1-P_\varepsilon)}\tau_\mathrm{r}. \tag{43}$$

The underdamped model is instead described by the UDL equations,
$$dx = \frac{1}{M}p\, dt, \qquad dp = -(Ax - b)\, dt - \frac{\gamma}{M}p\, dt + \mathcal{N}[0, 2\gamma\beta^{-1}\mathbb{I}dt]. \tag{44}$$
We define $\xi = \gamma/2M$, $\omega_j = \sqrt{\alpha_j/M}$, and $\zeta_j = \xi/\omega_j$. Moreover, a timescale $\tau_{\mathrm{r(UD)}}$ can be identified for the underdamped system which is analogous to the quantity τ_r associated with the overdamped system. In particular, we define $\tau_{\mathrm{r(UD)}} = \xi^{-1}$. We introduce a dimensionless quantity χ as well, which is $\chi = (1 + \xi/\omega_\mathrm{min})^{1/2}(1 - \xi/\omega_\mathrm{min})^{-1/2}$. With these definitions, we arrive at the following formulas for the timing parameters in the underdamped case:
$$\begin{aligned}\widehat{t_0} &= \max\left\{\tau_{\mathrm{r(UD)}}\ln\left(\kappa^{1/2}\chi\varepsilon_{\mu0}^{-1}\right), \frac{1}{2}\tau_{\mathrm{r(UD)}}\ln\left(\chi^2\kappa^{3/2}\varepsilon_{\Sigma0}^{-1}\left[\frac{1}{4\zeta_\mathrm{max}^2} + 1\right]\right)\right\}, \\ \widehat{\tau} &= \frac{2\sqrt{\kappa}\chi d\|A\|}{\beta\|b\|^2\varepsilon_x^2(1-P_\varepsilon)}\tau_{\mathrm{r(UD)}}.\end{aligned} \tag{45}$$

H Application: Matrix Inverse

H.0.1 Our Thermodynamic Algorithm

The linear systems protocol relies on estimating the mean of x, but make no use of the fluctuations in x at equilibrium. By using the second moments of the equilibrium distribution, we can go beyond solving linear systems. For example it is possible to find the inverse of a symmetric positive definite matrix A. As in the case of linear systems, the stationary distribution of x is $\mathcal{N}[A^{-1}b, \beta^{-1}A^{-1}]$. This means that the inverse of A can be obtained by evaluating the covariance matrix of x. This can be accomplished in an entirely analog way, using a combination of analog multipliers and integrators. By setting $b = 0$ for this protocol, we ensure that $\langle x\rangle = 0$, so the stationary covariance matrix is, by definition

$$\Sigma_{\rm s} = \lim_{t\to\infty} \langle x(t)x^{\intercal}(t)\rangle\,. \tag{46}$$

In order to estimate this, we again perform time averages after allowing the system to come to equilibrium

$$\Sigma_{\rm s} \approx \overline{xx^{\intercal}} = \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dt\, x(t)x^{\intercal}(t). \tag{47}$$

It is therefore necessary to have an analog component which evaluates the product $x_i(t)x_j(t)$ for each pair (i, j), resulting in d^2 analog multiplier components. Each of these products is then fed into an analog integrator component, which computes one element of the time-averaged covariance matrix

$$\Sigma_{{\rm s},ij} \approx \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dt\, x_i(t)x_j(t). \tag{48}$$

While the equilibration time is the same as for the linear system protocol, the integration time is different, because in general the covariance matrix is slower to converge than the mean. We now give a detailed description of the inverse estimation protocol, assuming ODL dynamics.

Inverse Estimation Protocol

1. Given a positive definite matrix A, set the potential of the device to

$$V(x) = \frac{1}{2}x^{\intercal}Ax \tag{49}$$

at time $t = 0$.

2. Choose equilibration tolerance parameter $\varepsilon_{\Sigma 0} \in \mathbb{R}^+$, and choose the equilibration time

$$t_0 \geq \widehat{t}_0, \tag{50}$$

where $\widehat{t}_0$ is computed from the system's physical properties, Eq. (53) or (54). Allow the system to evolve under its dynamics until $t = t_0$, which ensures that $\left\|\Sigma - \beta^{-1}A^{-1}b\right\| / \|\beta^{-1}A^{-1}\| \leq \varepsilon_{\Sigma}$.

3. Choose error tolerance parameter ε_{Σ} and success probability P_ε, and choose the integration time

$$\tau \geq \widehat{\tau}, \tag{51}$$

where $\widehat{\tau}$ is computed from the system's physical properties, Eq. (53) or (54). Use analog multipliers and integrators to measure the the time averages

$$\overline{x_ix_j} = \frac{1}{\tau^2}\int_{t_0}^{\tau} dt\, x_i(t)x_j(t), \tag{52}$$

which satisfies $\|\overline{xx^{\intercal}} - \beta^{-1}A^{-1}\|_F/\|\beta^{-1}A^{-1}\|_F \leq \varepsilon_A$ with probability at least P_ε.

The timing parameters for the inverse estimation protocol are, for the overdamped case,

$$\widehat{t}_0 = \frac{1}{2}\kappa\tau_{\rm r} \ln\left(2\kappa\varepsilon_{\Sigma 0}^{-1}\right), \quad \widehat{\tau} = \frac{4\kappa d(d+1)}{(1-P_\varepsilon)\varepsilon_{\Sigma}^2}\tau_{\rm r}, \tag{53}$$

and for the underdamped case

$$\widehat{t}_0 = \frac{1}{2}\tau_{\rm r(UD)} \ln\left(\chi^2\kappa^{3/2}\varepsilon_{\Sigma 0}^{-1}\left[\frac{1}{4\zeta_{\rm max}^2}+1\right]\right), \quad \widehat{\tau} = \frac{4\kappa d(d+1)}{(1-P_\varepsilon)\varepsilon_{\Sigma}^2}\tau_{\rm r(UD)}. \tag{54}$$

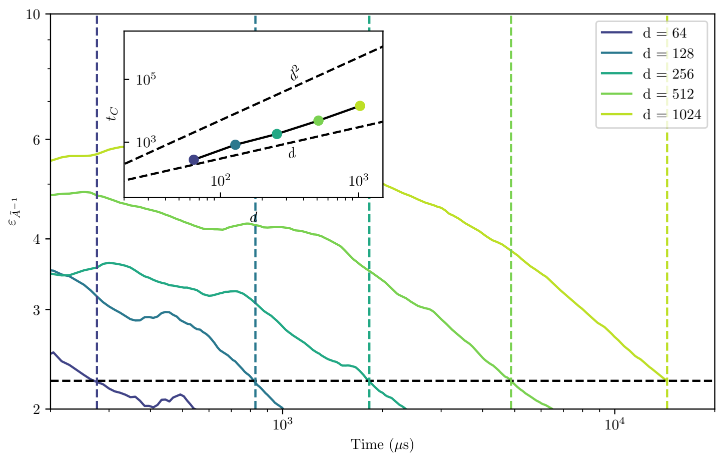

Figure 13: **Error of the inverse estimation thermodynamic algorithm as a function of the analog integration time for different dimensions.** Matrices A are drawn from a Wishart distribution with $2d$ degrees of freedom. Vertical dashed lines are the times t_C at which error goes below a threshold (horizontal dashed line). Inset: Crossing time t_C as a function of dimension d.

H.0.2 Comparison to digital methods

Similar to our analysis for the linear systems protocol, let us now examine the performance of the inverse estimation protocol in practical settings. To do so, we consider the error on the inverse, defined as

$$\varepsilon_{\tilde{A}} = \frac{\|\tilde{A} - A\|_F}{\|A\|_F}, \tag{55}$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

In Fig. 13, the convergence of the error as a function of the analog dynamics time for our thermodynamic inverse estimation algorithm is shown. We see that the expected convergence time to reach a given error is between linear and quadratic in the dimensionality of the system, in agreement with the analytical bounds presented in the previous section.

In addition, a runtime comparison to Cholesky decomposition was also performed, where a timing model similar to that employed for the linear systems protocol was used. The results are shown in Fig. 14, where the error is shown as a function of physical time for dimensions $100, 1000$ and 5000. The dashed lines represent the corresponding times for Cholesky decomposition, for given dimensions. We see that as the dimension grows, the advantage with respect to the Cholesky decomposition also grows, thus highlighting a practical thermodynamic advantage. Our method for the inverse estimation therefore has the advantage of having well-defined convergence properties as a function of dimension and condition number (compared to other approximate methods for inverting dense matrices, which do not have well defined convergence properties), as well as leading to reasonable error values in practical settings.

I Application: Solving Lyapunov Equations

In this section we assume we have access to a device with a controllable noise source such that the covariance matrix of the noise term may be chosen to be an arbitrary symmetric positive definite matrix. We do not include the linear $b^\intercal x$ term in the potential, and therefore obtain the following overdamped Langevin equation

$$dx = -\frac{1}{\gamma} Ax dt + \mathcal{N}\left[0, \frac{2}{\gamma\beta} R\, dt\right], \tag{56}$$

where R is symmetric and positive definite. In this case, the stationary distribution has mean zero and covariance matrix Σ_s, which is a solution to the Lyapunov equation

$$A\Sigma_\text{s} + \Sigma_\text{s} A^\intercal = 2\beta^{-1} R. \tag{57}$$

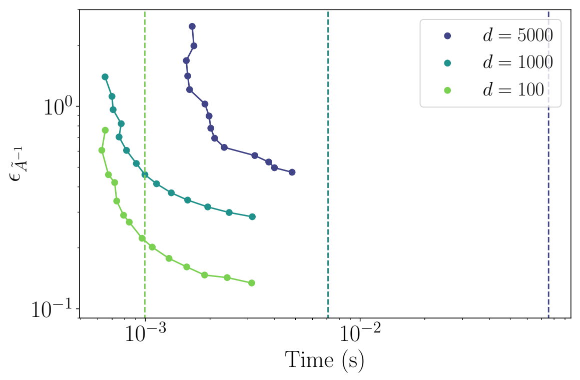

Figure 14: **Comparison of the error $\varepsilon_{\tilde{A}^{-1}}$ of the thermodynamic algorithm (TA) to solve linear systems with the Cholesky decomposition as a function of total runtime.** Dimensions are $d = 100, 1000, 5000$, respectively in light green, light blue, and purple are shown, as well as the corresponding Cholesky decomposition times as dashed lines. Here the condition numbers are respectively $\{120, 1189, 5995\}$. Calculations were performed on an Nvidia Tesla A10 GPU.

We propose the following protocol for solving the Lyapunov equation.

Lyapunov Equation Protocol

1. Given two symmetric positive definite matrices A and R, set the potential of the device to
$$V(x) = \frac{1}{2}x^\intercal Ax, \tag{58}$$
and the noise term in the overdamped Langevin equation to $\mathcal{N}\left[0, 2\gamma^{-1}\beta^{-1}Rdt\right]$ at time $t = 0$. That is, the system evolves under the dynamics of Eq. (56).
2. Choose equilibration tolerance parameter $\varepsilon_{\Sigma 0} \in \mathbb{R}^+$, and choose the equilibration time
$$t_0 \geq \widehat{t}_0, \tag{59}$$
where $\widehat{t}_0$ is computed from the system's physical properties, Eq. (53) or (54). Allow the system to evolve under its dynamics until $t = t_0$, which ensures that $\left\|\Sigma - \beta^{-1}A^{-1}b\right\| / \|\beta^{-1}A^{-1}\| \leq \varepsilon_\Sigma$.
3. Choose error tolerance parameter δ_Σ and success probability P_δ, and choose the integration time
$$\tau \geq \widehat{\tau}, \tag{60}$$
where $\widehat{\tau}$ is computed from the system's physical properties, Eq. (53) or (54). Use analog multipliers and integrators to measure the the time averages
$$\overline{x_i x_j} = \frac{1}{\tau^2}\int_{t_0}^{\tau} dt\, x_i(t)x_j(t), \tag{61}$$
which satisfies $\|\overline{xx^\intercal} - \Sigma_{\mathrm{s}}\|_F \leq \delta_\Sigma$ with probability at least P_δ.

The timing parameters for the Lyapunov equation protocol are, for the overdamped case,
$$\widehat{t}_0 = \frac{1}{2}\kappa\tau_{\mathrm{r}} \ln\left(\frac{\|\Sigma_0 - \Sigma_{\mathrm{s}}\| + \kappa\beta^{-1}\|A\|^{-1}}{\delta_{\Sigma 0}}\right), \quad \widehat{\tau} = \frac{4\kappa d(d+1)}{(1-P_\varepsilon)\varepsilon_\Sigma^2}\tau_{\mathrm{r}}, \tag{62}$$
For the underdamped case, $\widehat{t}_0$ would be somewhat different, but $\widehat{\tau}$ would be the same, because the behavior of the equilibrium correlation function does not depend on the noise, so the same result derived for the matrix inverse protocol is applicable. Note that Eq. (62) only vaguely determines the

equilibration time, as the target covariance matrix Σ_s is not known beforehand. The corresponding equilibration time $\widehat{t_0}$ for an underdamped system could also be evaluated in principle; however, this would only result in a similarly vague expression, which is anyway not necessary to determine the asymptotic time-complexity scaling of the algorithm, so it is not pursued here. Moreover, the relative error cannot be bounded as straightforwardly as was done for the other protocols given that there is no explicit formula for the target covariance matrix. For this reason, we have used absolute error as the error tolerance in the above protocol.

J Application: Matrix Determinant

The determinant of the covariance matrix appears in the normalization factor of a multivariate normal distribution, whose density function is

$$f_{\mu;\Sigma}(x) = (2\pi)^{-d/2} |\Sigma|^{-1/2} \exp\left(-\frac{1}{2} x^{\intercal} \Sigma^{-1} x\right), \tag{63}$$

and it is therefore natural to wonder whether hardware which is capable of preparing a Gaussian distribution may be used to somehow estimate the determinant of a matrix. This can in fact be done, as the problem is equivalent to the estimation of free energy differences, an important application of stochastic thermodynamics. Recall that the difference in free energy between equilibrium states of potentials V_1 and V_2 is [14]

$$\Delta F = F_2 - F_1 = -\beta^{-1} \ln\left(\frac{\int dx\, e^{-\beta V_2(x)}}{\int dx\, e^{-\beta V_1(x)}}\right). \tag{64}$$

Suppose the potentials are quadratic, with $V_1(x) = x^{\intercal} A_1 x$ and $V_2(x) = x^{\intercal} A_2 x$. Then each integral simplifies to the inverse of a Gaussian normalization factor,

$$\int dx\, e^{-\beta V_j(x)} = (2\pi)^{d/2} \sqrt{\beta^{-1} |A_j^{-1}|}, \tag{65}$$

so

$$\Delta F = -\beta^{-1} \ln\left(\sqrt{\frac{|A_2^{-1}|}{|A_1^{-1}|}}\right) = -\beta^{-1} \ln\left(\sqrt{\frac{|A_1|}{|A_2|}}\right). \tag{66}$$

This suggests that the determinant of a matrix A_1 can found by comparing the free energies of the equilibrium states with potentials V_1 and V_2 (where A_2 has known determinant), and then computing

$$|A_1| = e^{-2\beta\Delta F} |A_2|. \tag{67}$$

Fortunately, the free energy difference ΔF can be found, assuming we have the ability to measure the work which is done on the system as the potential $V(x)$ is changed from V_1 to V_2. According to the Jarzynski equality [34], the free energy difference between the (equilibrium) states in the initial and final potential is

$$e^{-\beta\Delta F} = \langle e^{-\beta W} \rangle, \tag{68}$$

where $\langle \cdot \rangle$ denotes an average over all possible trajectories of the system between time $t = 0$ and time $t = \tau$, weighed by their respective probabilities. This may be approximated by an average over N repeated trials,

$$e^{-\beta\Delta F} \approx \overline{e^{-\beta W}} \equiv \frac{1}{N} \sum_{j=1}^{N} e^{-\beta W_j}. \tag{69}$$

However, while Jarzynski's relation may be applied directly to estimate the free energy difference, this estimator has large bias and is slow to converge. Far more well-behaved estimators have been found based on work measurements. For simplicity, we here provide the expression based on Jarzynski's estimator, while Ref. [2] gives more suitable estimators. In summary, the determinant of A_1 is approximated by

$$|A_1| \approx \left(\overline{e^{-\beta W}}\right)^2 |A_2|. \tag{70}$$

In practice we will generally be interested in the log determinant to avoid computational overflow. This is

$$\ln(|A_1|) \approx 2 \ln\left(\overline{e^{-\beta W}}\right) + \ln(|A_2|). \tag{71}$$

We observe that, to estimate the log determinant to within (absolute) error δ_{LD} with probability at least P_δ, the total amount of time required is roughly

$$\tau \approx \frac{d\,\ln(\kappa)^2}{\delta_{\text{LD}}^2(1-P_\delta)} \ln\left(\chi^2\kappa^{3/2}\varepsilon_{\Sigma 0}^{-1}\left[\frac{1}{4\zeta_{\text{max}}^2}+1\right]\right)\tau_{\text{r(UD)}} = O(d\,\ln(\kappa)^3). \tag{72}$$