Supplement A: List of Acronyms

Acronym	Meaning
AI	Artificial Intelligence
ACP	Adaptive Correlation Penalty
DTM	Denoising Thermodynamic Model
DTCA	Denoising Thermodynamic Computer Architecture
EBM	Energy-Based Model
MEBM	Monolithic Energy-Based Model
FID	Fréchet Inception Distance
FLOP	Floating-Point Operation
GAN	Generative Adversarial Network
GPU	Graphics Processing Unit
HTDML	Hybrid Thermodynamic–Deterministic Machine Learning
LLM	Large Language Model
NN	Neural Network
PDK	Process Development Kit
RNG	Random-Number Generator
VAE	Variational Autoencoder
DDPM	Denoising Diffusion Probabilistic Model

Table S1. Acronyms used throughout this work.

Supplement B: Denoising Diffusion Models

Denoising diffusion models try to learn to time-reverse a random process that converts data into simple noise. Here, we will review some details on how these models work to support the analysis in the main text.

1. Forward Processes

The forward process is a random process that is used to convert the data distribution into noise. This conversion into noise is achieved through a stochastic differential equation in the continuous-variable case and a Markov jump process in the discrete case.

a. Continuous Variables

In the continuous case, the typical choice of forward process is the Itô diffusion,

$$dX(t) = -X(t)dt + \sqrt{2}\sigma dW \tag{1}$$

where $X(t)$ is a length N vector representing the state variable at time t, σ is a constant, and dW is a length N vector of independent Wiener processes.

The transition kernel for a random process defines how the probability distribution evolves in time,

$$Q_{t|0}(x'|x) = \mathbb{P}(X(t) = x'|X(0) = x) \tag{2}$$

For the case of Eq. (1) the transition kernel is,

$$Q_{t+s|s}(x'|x) \propto e^{-\frac{1}{2}(x'-\mu)^T\Sigma^{-1}(x'-\mu)} \tag{3}$$

$$\mu = e^{-t}x \tag{4}$$

$$\Sigma = \sigma^2 I \left(1 - e^{-2t}\right) \tag{5}$$

this solution can be verified by direct substitution into the corresponding Fokker-Planck equation. In the limit of infinite time, $\mu \to 0$ and $\Sigma \to \sigma^2 I$. Therefore, the stationary distribution of this process is zero-mean Gaussian noise with a standard deviation of σ.

b. Discrete Variables

The stochastic dynamics of some discrete variable X may be described by the Markov jump process,

$$\frac{dQ_t}{dt} = \mathcal{L} Q_t \tag{6}$$

where $\mathcal{L}$ is the generator of the dynamics, which is an $M \times M$ matrix that stores the transition rates between the various states. Q_t is a length M vector that assigns a probability to each possible state X may take at time t.

The transition rate from the i^{th} state to the j^{th} state is given by the matrix element $\mathcal{L}[j,i]$, which here takes the particular form,

$$\mathcal{L}[j,i] = \gamma\left(-(M-1)\delta_{j,i} + (1-\delta_{j,i})\right) \tag{7}$$

where δ is used to indicate the Kronecker delta function. Eq. (7) describes a random process where the probability per unit time to jump between any two states is γ.

Since Eq. (6) is linear, the dynamics of Q_t can be understood entirely via the eigenvalues and eigenvectors of $\mathcal{L}$,

$$\mathcal{L} v_k = \lambda_k v_k \tag{8}$$

Note that by symmetry of $\mathcal{L}$, we do not distinguish between right and left eigenvectors.

One eigenvector-eigenvalue pair $(v_0, \lambda_0 = 0)$ corresponds to the unique stationary state of $\mathcal{L}$, with all entries of v_0 being equal to some constant (if normalized, then $v_0[j] = \frac{1}{M}$ for all j). The long-time dynamics of this MJP transform any initial distribution to a uniform distribution over all states.

The remaining eigenvectors are decaying modes associated with negative eigenvalues. These additional $M-1$ eigenvectors take the form,

$$v_j[i] = -\delta_{i,0} + \delta_{i,j} \tag{9}$$

$$\lambda_j = -\gamma M \tag{10}$$

where Eq. (9) and Eq. (10) are valid for $j \in [1, M-1]$. Therefore, all solutions to this MJP decay exponentially to the uniform distribution with rate γM.

The time-evolution of Q is given by the matrix exponential,

$$Q_t = e^{\mathcal{L}t} Q_0. \tag{11}$$

This matrix exponential is evaluated by diagonalizing $\mathcal{L}$,

$$e^{\mathcal{L}t} = P e^{Dt} P^{-1} \tag{12}$$

where the columns of P are the M eigenvectors v_k and D is a diagonal matrix of the eigenvalues λ_k.

Using the solution for the eigenvalues and eigenvectors found above, we can solve for the matrix elements of $e^{\mathcal{L}t}$,

$$e^{\mathcal{L}t}[j,i] = \delta_{i,j}\left(\frac{1 + (M-1)e^{-\gamma M t}}{M}\right) + (1-\delta_{i,j})\left(\frac{1 - e^{-\gamma M t}}{M}\right) \tag{13}$$

Using this solution, we can deduce an exponential form for the matrix elements of $e^{\mathcal{L}t}$,

$$e^{\mathcal{L}t}[j,i] = \frac{1}{Z(t)} e^{\Gamma(t)\delta_{i,j}} \tag{14}$$

$$\Gamma(t) = \ln\left(\frac{1 + (M-1)e^{-\gamma t}}{1 - e^{-\gamma t}}\right) \tag{15}$$

$$Z(t) = \frac{M}{1 - e^{-\gamma t}} \tag{16}$$

Now consider a process in which each element of the vector of N discrete variables X undergoes the dynamics described by Eq. (6) independently. In that case, the differential equation describing the dynamics of the joint distribution Q_t is,

$$\frac{dQ_t}{dt} = \sum_{k=1}^{N} \left(I_1 \otimes \cdots \otimes \mathcal{L}_k \otimes \ldots I_N\right) Q_t \tag{17}$$

where I_j indicates the identity operator and $\mathcal{L}_j$ the operator from Eq. (7) acting on the subspace of the j^{th} discrete variable.

The Kronecker product of the matrix exponentials gives the time-evolution of the joint distribution,

$$e^{\mathcal{L}t} = \bigotimes_{k=1}^{N} e^{\mathcal{L}_k t} \tag{18}$$

with the matrix elements,

$$e^{\mathcal{L}t}[j, i] = \prod_{k=1}^{N} e^{\mathcal{L}_k t}[j_k, i_k] \tag{19}$$

where j and i are now vectors with N elements, indexed as i_k or j_k respectively.

Using Eqs. (14) - (16), we can find an exponential form for the joint process transition kernel (as defined in Eq. (2)),

$$Q_{t|0}(x'|x) = \frac{1}{Z(t)} \exp\left(\sum_{k=1}^{N} \Gamma_k(t)\delta_{x'[k],x[k]}\right) \tag{20}$$

$$Z(t) = \prod_{k=1}^{N} Z_k(t) \tag{21}$$

$\Gamma_k(t)$ and $Z_k(t)$ are as given in Eqs. (15) and (16), with each dimension potentially having it's own transition rate γ_k and number of categories M_k.

2. Reverse Processes

In general, a random process for some variable X can be reversed using Bayes' rule,

$$Q_{t|t+\Delta t}(x'|x) = \frac{Q_{t+\Delta t|t}(x|x')Q_t(x')}{Q_{t+\Delta t}(x)} \tag{22}$$

where the conditionals are as defined in Eq. (2), and the marginals are,

$$Q_t(x) = \mathbb{P}\left(X(t) = x\right) \tag{23}$$

A differential equation that describes the reverse process can be found by analyzing Eq. (22) in the infinitesimal time limit. Specifically, defining a reversed time $t = T - s$ given some arbitrary endpoint T and expanding Eq. (22) in Δs,

$$Q_{T-s|T-(s-\Delta s)}(x'|x) \approx \delta_{x,x'} + \Delta s\, \mathcal{L}_{\text{rev}}(x', x) \tag{24}$$

where $\mathcal{L}_{\text{rev}}$ is the generator of the reverse process,

$$\mathcal{L}_{\text{rev}}(x', x) = \frac{Q_{T-s}(x')}{Q_{T-s}(x)} \lim_{\Delta s \to 0} \left[\frac{d}{d\Delta s} Q_{T-(s-\Delta s)|T-s}(x|x') \right] + \delta_{x,x'} \left(\frac{1}{Q_{T-s}(x)} \frac{dQ_{T-s}(x)}{ds} \right) \tag{25}$$

here, $\delta_{x,x'}$ is used to indicate the Dirac delta function in the continuous case and the Kronecker delta in the discrete case.

If the dynamics of Q are linear and generated by $\mathcal{L}$ like Eq. (6), we can simplify,

$$\lim_{\Delta s \to 0} \left[\frac{d}{d\Delta s} Q_{T-(s-\Delta s)|T-s}(x|x') \right] = \mathcal{L}(x, x') \tag{26}$$

In this case, we can re-write Eq. (25) in the operator form,

$$\mathcal{L}_{\text{rev}} = Q \mathcal{L}^\dagger Q^{-1} + Q^{-1} \frac{dQ}{ds} \tag{27}$$

where $\mathcal{L}^\dagger$ is the adjoint operator to $\mathcal{L}$. For continuous variables, the adjoint operator is defined as,

$$\int \psi_2 \mathcal{L} \psi_1 dx = \int \psi_1 \mathcal{L}^\dagger \psi_2 dx \tag{28}$$

for any test functions ψ_1 and ψ_2. For discrete variables, $\mathcal{L}^\dagger = \mathcal{L}^T$.

a. Continuous variables

In the case that the forward process is an Itô diffusion, $\mathcal{L}$ is the generator for the corresponding Fokker-Planck equation,

$$\mathcal{L} = -\sum_i \frac{\partial}{\partial x_i} f_i(x, t) + \frac{1}{2} \sum_{i,j} \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} D_{ij}(t) \tag{29}$$

where D is a symmetric matrix, $D_{ij} = D_{ji}$ that does not depend on x.

Using Eq. (28) and integration by parts, it can be shown that the adjoint operator is,

$$\mathcal{L}^\dagger = \sum_i f_i \frac{\partial}{\partial x_i} + \frac{1}{2} \sum_{i,j} D_{ij} \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_i} \tag{30}$$

By directly substituting Eq. (30) into Eq. (27) and simplifying, $\mathcal{L}_{\text{rev}}$ can be reduced to,

$$\mathcal{L}_{\text{rev}} = \sum_i \frac{\partial}{\partial x_i} g_i + \frac{1}{2} \sum_{i,j} \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} D_{ij} \tag{31}$$

with the drift vector g,

$$g_i(x, t) = f_i(x, t) - \frac{1}{Q_t(x)} \sum_j \frac{\partial}{\partial x_j} \left[D_{ij}(x, t) Q_t(x) \right] \tag{32}$$

If Δt is chosen to be sufficiently small, Eq. (32) can be linearized and the transition kernel is Gaussian,

$$Q_{t|t+\Delta t}(x'|x) \propto \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right) \tag{33}$$

$$\mu = x + \Delta t \, g_i(x, t) \tag{34}$$

$$\Sigma = \Delta t \, D(t) \tag{35}$$

Therefore, one can build a continuous diffusion model with arbitrary approximation power by working in the small Δt limit and approximating the reverse process using a Gaussian distribution with a neural network defining the mean vector [1, 2].

b. *Discrete variables*

In a discrete diffusion model, $\mathcal{L}$ is given by Eq. (17). This tensor product form for $\mathcal{L}$ guarantees that $\mathcal{L}(x', x) = 0$ for any vectors x' and x that have a Hamming distance greater than one (which means they have at least $N - 1$ matching elements). As such, in discrete diffusion models, neural networks trained to approximate ratios of the data distribution $\frac{Q_{T-s}(x')}{Q_{T-s}(x)}$ for neighboring x' and x can be used to implement an arbitrarily good approximation to the actual reverse process [3].

3. The Diffusion Loss

As discussed in the main text, a diffusion model is trained by minimizing the distributional distance between the joint distributions of the forward process $Q_{0,\ldots,T}$ and our learned approximation to the reverse process $P^{\theta}_{0,\ldots,T}$,

$$\mathcal{L}_{DN}(\theta) = D\left(Q_{0,\ldots,T}(\cdot)||P^{\theta}_{0,\ldots,T}(\cdot)\right) \tag{36}$$

the Markovian nature of Q can be taken advantage of to simplify Eq. (36) into a layerwise form,

$$\mathcal{L}_{DN}(\theta) + C = -\sum_{t=1}^{T} \mathbb{E}_{Q(x_{t-1},x_t)}\left[\log\left(P_\theta(x_{t-1}|x_t)\right]\right. \tag{37}$$

where C does not depend on θ. For denoising algorithms that operate in the infinitesimal limit, the simple form of P_θ allows for $\mathcal{L}_{DN}$ and its gradients to be computed exactly.

a. *A Monte-Carlo gradient estimator*

In the case where $P_\theta\left(x^{t-1}|x^t\right)$ is an EBM, there exists no simple closed-form expression for $\nabla_\theta \mathcal{L}_{DN}(\theta)$. In that case, one must employ a Monte Carlo estimator to approximate the gradient. This estimator can be derived directly by taking the gradient of Eq. (37),

$$\nabla_\theta \mathcal{L}_{DN}(\theta) = -\sum_{t=1}^{T} \mathbb{E}_{Q(x^{t-1},x^t)}\left[\nabla_\theta \log\left(P_\theta(x^{t-1}|x^t)\right)\right] \tag{38}$$

If we have an EBM parameterization for $P_\theta\left(x^{t-1}|x^t\right)$ this may be simplified further. Specifically, given the latent variable from Eq. 8 in the main text, the gradient of log-likelihood may be simplified to,

$$\nabla_\theta \log\left(P_\theta(x^{t-1}|x^t)\right) = \mathbb{E}_{P_\theta(x^{t-1},z^{t-1}|x^t)}\left[\nabla_\theta \mathcal{E}^m_{t-1}\right] - \mathbb{E}_{P_\theta(z^{t-1}|x^{t-1},x^t)}\left[\nabla_\theta \mathcal{E}^m_{t-1}\right] \tag{39}$$

Inserting this into Eq. (38) yields the final result given in Eq. 14 in the article.

4. Simplification of the Energy Landscape

As the forward process timestep is made smaller, the energy landscape of the EBM-based approximation to the reverse process becomes simpler. A simple 1D example serves as a good demonstration of this concept. Consider the marginal energy function,

$$\mathcal{E}^{\theta}_{t-1}\left(x_{t-1}\right) = \left(x^2_{t-1} - 1\right)^2 \tag{40}$$

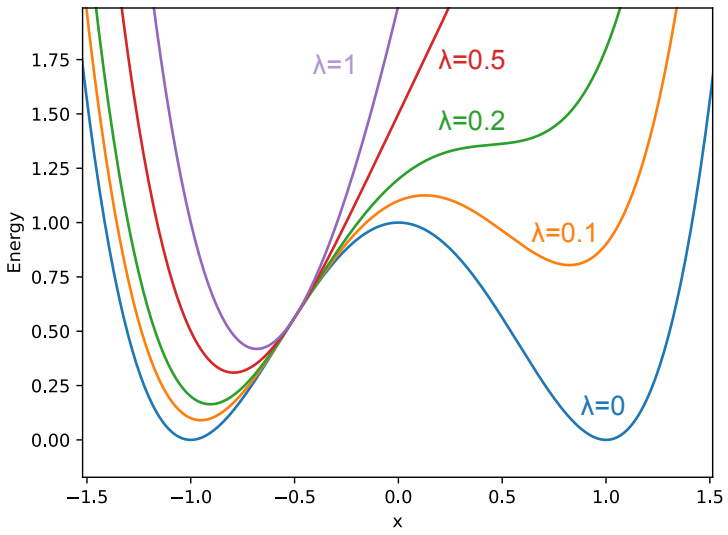

Figure S1. **Conditioning of the energy landscape** As λ is increased, the energy landscape is reshaped from a strongly bimodal distribution towards a simple Gaussian centered at $x_t = -0.5$. The latter is much easier to sample from.

and a forward process energy function that corresponds to Gaussian diffusion (Eq. (1)),

$$\mathcal{E}^f_{t-1}(x_{t-1}, x_t) = \lambda \left(\frac{x_{t-1}}{x_t} - 1 \right)^2 \tag{41}$$

The parameter λ scales inversely with the size of the forward process timestep; that is, $\lim_{\Delta t \to 0} \lambda = \infty$.

The reverse process conditional energy landscape is then $\mathcal{E}^\theta_{t-1} + \mathcal{E}^f_{t-1}$. The effect of λ on this is shown in Fig. 1.

The energy landscape is bimodal at $\lambda = 0$ and gradually becomes distorted towards an unimodal distribution centered at x_t as λ increases. This reshaping is intuitive, as shortening the forward process timestep should more strongly constrain x_{t-1} to x_t.

5. Conditional Generation

The denoising framework can be adapted for conditional generation tasks, such as generating MNIST digits given a specific class label. In principle, this is very simple: we concatenate the target (in our case, the images) and a one-hot encoding of the labels into a contiguous binary vector and treat that whole thing as our training data on which we train the denoising model as described above.

In this case, the visible nodes of the Boltzmann machine are partitioned into "pixel nodes" V_X and "label nodes" V_L. All visible nodes come in pairs of input and output nodes (drawn in blue and green resp. in Fig. 3 in the main paper body and Fig. 3 below), so the set of visible nodes now consists of $V_X^{\text{in}}, V_X^{\text{out}}, V_L^{\text{in}}$, and V_L^{out}.

The training procedure works the same way as before, just using this label-augmented data. We obtain the noised training images X_n and labels L_n by noising each entry of X_0 and L_0 resp. independently using the forward process described in subsection B 1 b. Then we train the nth step model $P_{\theta_n}\big(V_X^{\text{out}} = x, V_L^{\text{out}} = l \,|\, V_X^{\text{in}} = x', V_L^{\text{in}} = l'\big)$ to approximate (in terms Kullback-Leibler divergence) the distribution $\mathbb{P}\big(X_n = x, L_n = l \,|\, X_{n+1} = x', L_{n+1} = l'\big)$.

At inference time, we have two cases:

- Unconditional inference proceeds as with regular denoising. We pass the pixel and label values backward through all the step models, and at the end, we record the pixel values.
- For conditional generation we clamp all label output nodes V_L^{out} in all step models to l_0 and sample $\widehat{X}_n \sim P_{\theta_n}\big(V_X^{\text{out}} = \cdot \,|\, V_X^{\text{in}} = \widehat{X}_{n+1}, V_L^{\text{out}} = V_L^{\text{in}} = l_0\big)$, where l_0 is an unnoised label and $\widehat{X}_{n+1}$ is the output of $P_{\theta_{n+1}}$ (or uniform noise if $n = N$).

Note that all step models except θ_0 will be trained on somewhat noised labels, so they might never have seen a pristine unnoised label during training (if there are 10 classes and five label repetitions, a strongly noised label has an approximately 10×2^{-50} chance of being a valid unnoised label). However, during conditional inference, the models will have their label nodes clamped to an unnoised label l_0, and they may not know how this should influence the generated image (and this problem would only be exacerbated if we clamped to a noised label instead).

This issue can be mitigated by using a rate γ_X when noising image entries in the training data and a different rate γ_L for noising label entries. Recall that the higher the γ, the noisier the data will become as n increases.

We consider two extremes:

- If $\gamma_L \geq \gamma_X$, then we have the exact same problem as before.
- If $\gamma_L = 0$, then the labels in the training data are a zero-temperature distribution. This low temperature can lead to freezing, potentially negating the benefits denoising could otherwise bring.

Experimentally, we observed that settings in the ranges $\gamma_L \in [0.1, 0.3]$ and $\gamma_X \in [0.7, 1.5]$ (for models with four to 12 steps) yielded good conditional generation performance while avoiding the freezing problem.

6. Learning the marginal

If a DTM is trained to match the conditional distribution of the reverse process perfectly, the learned energy function $\mathcal{E}^\theta_{t-1}$ is the energy function of the true marginal distribution, that is, $\mathcal{E}^\theta_{t-1}(x) \propto \log Q(x^{t-1})$. To show this, we start by applying the Bayes' rule to the learned reverse process conditional in the limit that it perfectly matches the true reverse process,

$$\frac{Q(x^t|x^{t-1})Q(x^{t-1})}{Q(x^t)} = \frac{1}{Z(\theta, x^t)} e^{-\left(\mathcal{E}^f_{t-1}\left(x^{t-1}, x^t\right) + \mathcal{E}^\theta_{t-1}\left(x^{t-1}, \theta\right)\right)} \tag{42}$$

defining the distribution,

$$H(x^{t-1}) = \frac{1}{Z(\theta)} \sum_{z^{t-1}} e^{-\mathcal{E}^\theta_{t-1}\left(x^{t-1}, z^{t-1}, \theta\right)} \tag{43}$$

$$Z(\theta) = \sum_{x^{t-1}, z^{t-1}} e^{-\mathcal{E}^\theta_{t-1}\left(x^{t-1}, z^{t-1}, \theta\right)} \tag{44}$$

extracting the forward process from the RHS of Eq. (42) and using Eq. (43),

$$\frac{Q(x^{t-1})}{Q(x^t)} = \frac{Z(\theta)Z}{Z(\theta, x^t)} H(x^{t-1}) \tag{45}$$

Eq. (45) can easily be re-arranged into a form where the LHS depends only on x^t, and the RHS depends only on x^{t-1}. From this, we can deduce,

$$\frac{Q(x^{t-1})}{H(x^{t-1})} = c \tag{46}$$

from the fact that Q and H are both normalized, we can find that $c = 1$, which establishes the desired equivalence.

Supplement C: Hardware accelerators for EBMs

In this work, we focus on a hardware architecture for EBMs that are naturally expressed as Probabilistic Graphical Models (PGMs). In a PGM-EBM, the random variables involved in the model map to the nodes of a graph, which are connected by edges that indicate dependence between variables.

PGMs form a natural basis for a hardware architecture because they can be sampled using a modular procedure that respects the graph's structure. Specifically, the state of a PGM can be updated by iteratively stepping through each node of the graph and resampling one variable at a time, using only information about the current node and

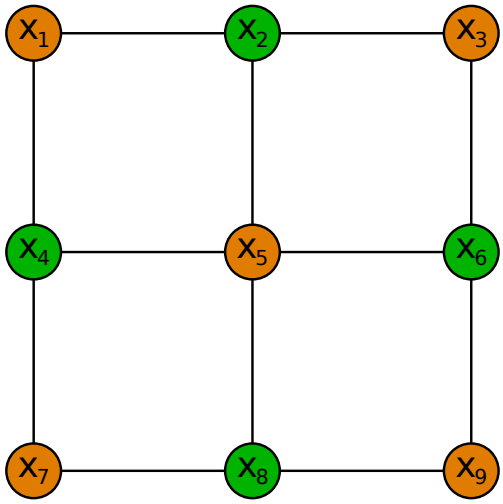

Figure S2. **Chromatic Gibbs Sampling** A schematic view of an abstract hardware accelerator for a simple EBM. Each of the model's variables is assigned to a node. Each node is capable of receiving information from its neighbors and updating its state according to the appropriate conditional distribution. Since each node's update distribution only depends on the state of its neighbors and because nodes of the same color do not neighbor each other, they can all be updated in parallel.

its immediate neighbors. Therefore, if a PGM is local, sparse, and somewhat heterogeneous, a piece of hardware can be built to efficiently sample from it that involves spatially arraying probabilistic sampling circuits that interact with each other cheaply via short wires.

This local PGM sampler represents a type of compute-in-memory approach, where the state of the sampling program is spatially distributed throughout the array of sampling circuitry. Since the sampling circuits only communicate locally, this type of computer will spend significantly less energy on communication than one built on a Von-Neumann-like architecture, which constantly shuttles data between compute and memory.

Formally, the algorithm that defines this modular sampling procedure for PGMs is called Gibbs sampling. In Gibbs sampling, samples are drawn from the joint distribution $p(x_1, x_2, \ldots, x_N)$ by iteratively updating the state of each node conditioned on the current state of its neighbors. For the i^{th} node, this means sampling from the distribution,

$$x_i[t+1] \sim p(x_i|nb(x_i)[t]). \tag{47}$$

This procedure defines a Markov chain whose stationary distribution can be easily controlled by adjusting the conditional update distributions of each node (see the next section for an example). Starting from some random initialization, this iterative update must be applied potentially many times to all graph nodes before the Markov chain converges to the desired stationary distribution, allowing us to draw samples from it.

Gibbs sampling allows for any two nodes that are not neighbors to be updated in parallel, meaning that the state can be updated in batches corresponding to different color groups of the graph. For a more thorough explanation of how Gibbs sampling works, see [4].

Fig. 2 shows a simple example of a PGM with two color groups that would be amenable to Gibbs sampling. Since x_1 is only connected to x_2 and x_4, the update rule from Eq. (47) would take the form,

$$x_1[t+1] \sim p(x_1|x_4[t], x_2[t]) \tag{48}$$

If the joint distribution had sufficient structure such that the conditional for each node had the same form, a piece of hardware could be built to sample from this PGM by building a 3x3 grid of sampling circuits that communicate only with their immediate neighbors.

1. Quadratic EBMs

The primary constraint around building a hardware device that implements Gibbs sampling is that the conditional update given in Eq. (47) must be efficiently implementable. Generally, this means that one wants it to take a form that is "natural" to the hardware substrate being used to build the computer.

To satisfy this constraint, it is generally necessary to limit the types of joint distributions that a hardware device can sample from. An example of such a restricted family of distributions is quadratic EBMs.

Quadratic EBMs have energy functions that are quadratic in the model's variables, which generally leads to conditional updates computed by biasing a simple sampling circuit (Bernoulli, categorical, Gaussian, etc.) with the output of a linear function of the neighbor states and the model parameters. These simple interactions are efficient to implement in various types of hardware. As such, Quadratic EBMs have been the focus of most work on hardware accelerators for Gibbs sampling to date.

In the main text, we discuss Boltzmann machines, which involve only binary random variables and are the most basic form of quadratic EBM. The Conditional Update for Boltzmann Machines requires biasing a Bernoulli random variable according to a sigmoid function of a linear combination of the model parameters and the binary neighbor states, as shown in the main text, Eq. 11. This conditional update is efficiently implementable using an RNG with a sigmoidal bias and resistors, as discussed in section K.

Here, we will touch on a few other types of quadratic EBM that are more general. Although the experiments in this paper focused on Boltzmann machines, they could be trivially extended to these more expressive classes of distributions.

a. Potts models

Potts models generalize the concept of Boltzmann machines to k-state variables. They have the energy function,

$$E(x) = \sum_{i,j=1}^{N} \sum_{m,n=1}^{M} x_m^i J_{mn}^{ij} x_n^j + \sum_{i=1}^{N} \sum_{m=1}^{M} h_m^i x_m^i \tag{49}$$

$$J_{mn}^{ii} = 0 \tag{50}$$

x_m^i is a one-hot encoding of the state of variable x^i,

$$x_m^i \in \{0, 1\} \tag{51}$$

$$\sum_m x_m^i = 1 \tag{52}$$

which implies that $x_m^i = 1$ for a single value of m, and is zero otherwise. The distribution of any individual variable conditioned on it's Markov blanket is,

$$p(x_m^i = 1 | \text{mb}(x^i)) = \frac{1}{Z} \exp\left(-\beta \left(\sum_{j \in \text{mb}(x^i), n} J_{mn}^{ij} x_n^j + \sum_{j \in \text{mb}(x^i), n} x_n^j J_{nm}^{ji} + h_m^i \right)\right) \tag{53}$$

In the case that J has the symmetry,

$$J_{mn}^{ij} = J_{nm}^{ji} \tag{54}$$

this reduces to,

$$p(x_m^i = 1 | \text{mb}(x^i)) \propto \frac{1}{Z} e^{-\theta_m^i} \tag{55}$$

$$\theta_m^i = \beta \left(2 \sum_{j \in \mathrm{mb}(x^i), n} J_{mn}^{ij} x_n^j + h_m^i \right) \tag{56}$$

The parameters θ are defined to make it clear that this is a *softmax* distribution.

Therefore, to build a hardware device that samples from Potts models using Gibbs sampling, one would have to build a softmax sampling circuit parameterized by a linear function of the model weights and neighbor states. Potts model sampling is slightly more complicated than Boltzmann machine sampling, but it is likely possible.

b. *Gaussian-Bernoulli EBMs*

Gaussian-Bernoulli EBMs extend Boltzmann machines to continuous, binary mixtures. In general, this type of model can have continuous-continuous, binary-binary, and binary-continuous interactions. For simplicity, if we consider only binary-continuous interactions, the energy function may be written as,

$$E(v, h) = \sum_{i=1}^{N_v} \frac{(v_i - b_i)^2}{2\sigma_i^2} - \sum_{i=1}^{N_v} \sum_{j=1}^{N_h} \frac{v_i W_{ij} h_j}{\sigma_i^2} - \sum_{j=1}^{N_h} c_j h_j, \tag{57}$$

where $v_i \in \mathbb{R}$ are continuous variables with biases b_i and variances σ_i^2, $h_j \in \{-1, 1\}$ are binary variables with biases c_j, and W_{ij} are interaction weights.

Due to the structure of the energy function, the update rule for the continuous variables corresponds to drawing a sample from a Gaussian distribution with a mean that is a linear function of the neighbor states,

$$p(v_i \,|\, \mathrm{mb}(v_i)) = \mathcal{N}\big(\mu_i, \sigma_i^2/\beta\big), \qquad \mu_i = b_i + \sigma_i^2 \sum_{j \in \mathrm{mb}(v_i)} W_{ij} h_j. \tag{58}$$

The binary update rule is similar to the rule for Boltzmann machines,

$$p(h_j = 1 \,|\, \mathrm{mb}(h_j)) = \sigma \left(2\beta \left(\sum_{i \in \mathrm{mb}(h_j)} \frac{v_i W_{ij}}{\sigma_i^2} + c_j \right) \right) \tag{59}$$

Hardware implementations of Gaussian-Bernoulli EBMs are more difficult than the strictly discrete models because the signals being passed during conditional sampling of the binary variables are continuous. To pass these continuous values, they must either be embedded into several discrete variables or an analog signaling system must be used. Both of these solutions would incur significant overhead compared to the purely discrete models.

Supplement D: A hardware architecture for denoising

The denoising models used in this work exclusively modeled distributions of binary variables. The reverse process energy function (Eq. 7 in the main text) was implemented using a Boltzmann machine. The forward process energy function $\mathcal{E}_{t-1}^f$ was implemented using a simple set of pairwise couplings between x^t (blue nodes) and x^{t-1} (green nodes). The marginal energy function $\mathcal{E}_{t-1}^\theta$ was implemented using a latent variable model (latent nodes are drawn in orange) with a sparse, local coupling structure.

1. Implementation of the forward process energy function

From the exponential form of the discrete-variable forward process transition kernel given in Eq. (20), it is straightforward to derive a Boltzmann machine-style energy function that implements the forward process,

$$\mathcal{E}_{t-1}^f = \sum_i \frac{\Gamma_i(t)}{2} x_i^t x_i^{t-1} \tag{60}$$

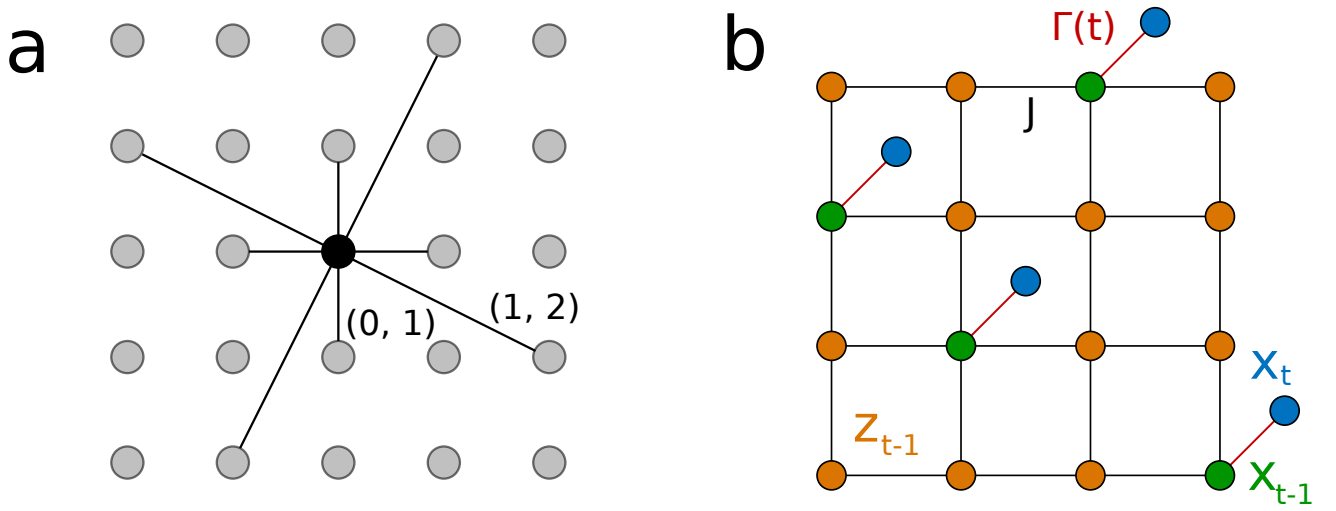

Figure S3. **Our hardware denoising architecture (a)** An example of a possible connectivity pattern as specified in Table. 2. For clarity, the pattern is illustrated as applied to a single cell; however, in reality, the pattern is repeated for every cell in the grid. **(b)** A graph for hardware denoising. The grid is subdivided at random into visible (green) nodes, representing the variables x^{t-1}, and latent (orange) nodes, representing z^{t-1}. Each visible node x_j^{t-1} is coupled to a (blue) node carrying the value from the previous step of denoising x_j^t (note that these blue nodes stay fixed throughout the Gibbs sampling).

Pattern	Connectivity
G_8	$(0,1),\ (4,1)$
G_{12}	$(0,1),\ (4,1),\ (9,10)$
G_{16}	$(0,1),\ (4,1),\ (8,7),\ (14,9)$
G_{20}	$(0,1),\ (4,1),\ (3,6),\ (8,7),\ (14,9)$
G_{24}	$(0,1),\ (1,2),\ (4,1),\ (3,6),\ (8,7),\ (14,9)$

Table S2. Edges (ordered pairs) associated with graphs of various degrees.

where $x_t[i] \in \{-1, 1\}$ indicates the i^{th} element of the vector of random variables x_t as usual.

2. Implementation of the marginal energy function

We use a Boltzmann machine based on a grid graph to implement the marginal energy function. Our grids have both nearest-neighbor and long-range skip connections. A simple example of this is shown in Fig. 3 (a). This connectivity pattern is tiled such that every node in the bulk of the grid has the same connectivity to its neighbors. At the boundaries, connections that extend beyond the grid's edges are not formed.

Within the grid, we randomly choose some subset of the nodes to represent the data variables x_{t-1}. The remaining nodes then implement the latent variable z_{t-1}. The grid is, therefore, a deep Boltzmann machine with a sparse connectivity structure and multiple hidden layers.

We use a particular set of connectivity patterns in the experiments in this article, which are specified in Table. 2. We say that node (x, y) has a connection rule of the form (a, b) if it is connected to nodes at positions $(x + a, y + b), (x - b, y + a), (x - a, y - b), (x + b, y - a)$, so each connection rule adds up to 4 edges from this node.

As explicitly stated in Eq. 7 of the article, our variational approximation to the reverse process conditional has an energy function that is the sum of the forward process energy function and the marginal energy function. Physically, this corresponds to adding nodes to our grid that implement x_t, which are connected pairwise to the data nodes implementing x_{t-1} via the coupling defined in Eq. (60). This connectivity is shown in Fig. 3 (b).

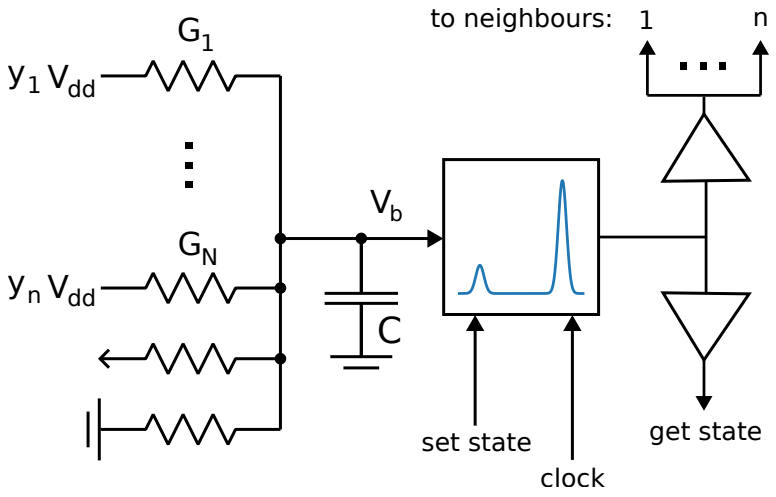

Figure S4. **A schematic of a possible Boltzmann machine sampling cell** A linear resistor network computes a biasing voltage given the sign-corrected neighbor states $y_n = x_n \oplus s_n$. The output of this circuit biases an RNG that responds in a sigmoidal manner. This RNG processes freely when the clock is low and latches to a state when the clock is high. Upon the clock going high, the sampled state is broadcasted to the neighbors of the cell over wires.

Supplement E: Energetic analysis of the hardware architecture

Our RNG design uses only transistors and can integrate tightly with other traditional circuit components on a chip to implement a large-scale sampling system. Since there are no exotic components involved that introduce unknown integration barriers, it is straightforward to build a simple physical model to predict how this device utilizes energy.

The performance of the device can be understood by analyzing the unit sampling cell that lives on each node of the PGM implemented by the hardware. The function of this cell is to implement the Boltzmann machine conditional update, as given in Eq. 11 in the main text.

There are many possible designs for the sampling cell. The design considered here utilizes a linear analog circuit to combine the neighboring states and model weights, producing a control voltage for an RNG. This RNG then produces a random bit that is biased by a sigmoidal function of the control voltage. This updated state is then broadcast back to the neighbors. The cell must also support initialization and readout (get/set state operations). A schematic of a unit cell is shown in Fig. 4.

We provide experimental measurements of our novel RNG circuitry in the main text, which establish that random bits can be produced at a rate of $\tau_{rng}^{-1} \approx 10\,\text{MHz}$ using $\sim 350\text{aJ}$ of energy per bit. Fig. 9 (a) shows an output voltage waveform from the RNG circuit. It wanders randomly between high and low states. Critically, the bias of the RNG circuit (the probability of finding it in the high or low state) is a sigmoidal function of its control voltage, which allows for a straightforward implementation of the conditional update using linear circuitry.

The size of the RNG circuit can be used to anchor the dimensions of a future large-scale Gibbs sampling device. As shown in Fig. 9 (b), the RNG itself involves around 10 transistors and takes up $\sim 3\mu m \times 3\mu m$ on the die. It is reasonable to imagine that the whole sampling cell could fit in $4\times$ this area and have a side length of $6\mu m$. Given this area, a 1000×1000 grid of sampling cells would fit within a 6mm $\times$ 6mm chip.

Building on the measured characteristics of our RNG, we will now develop simple physical models for the remaining components of the sampling system. These models can then be combined to estimate the energy consumption of the diffusion models developed in this article running on our hardware.

1. Biasing circuit

The multiply-accumulation of the model weights and neighbor states can be performed using a resistor network, as shown in Fig. 4. The dynamics of this resistor network are described by the differential equation,

$$\sum_{j=1}^{n+2} G_j \left(V_{dd}\, y_j - V_b\right) = C\frac{dV_b}{dt} \tag{61}$$

where $y_i = x_i \oplus s_i$ is the XOR of the neighbor state x_i with a sign bit s_i. There are n variable neighbor states and two fixed inputs ($y_{n+1} = 1$, $y_{n+2} = 0$), which are important for implementing the fixed bias term in the conditional update. V_{dd} is the supply voltage and V_b is the output voltage that biases the RNG. G_i represents the conductance of the resistor corresponding to the i^{th} input. The capacitance C represents the parasitic capacitance to ground associated with any real implementation of this circuit and is critical to forming realistic estimates of speed and energy consumption. Realistic values for an implementation of this circuit in our transistor process are shown in Fig. 5 (a).

Since this equation is first order, the dynamics exponentially relax to some fixed point V_b^∞,

$$V_b(t) = c\, e^{-t/\tau_{bias}} + V_b^\infty \tag{62}$$

the time constant τ_{bias} is,

$$\tau_{bias} = \frac{C}{G_\Sigma} \tag{63}$$

and the fixed point is,

$$V_b^\infty = \sum_{j=1}^{n+2} \frac{G_j}{G_\Sigma} V_{dd} y_j \tag{64}$$

where the total conductance G_Σ is,

$$G_\Sigma = \sum_{j=1}^{n+2} G_j \tag{65}$$

The RNG has a bias curve which takes the form,

$$\mathbb{P}\left(x_i = 1\right) = \sigma\left(\frac{V_b}{V_s} - \phi\right) \tag{66}$$

inserting Eq. (64) and expanding the term inside the sigmoid,

$$\frac{V_b}{V_s} - \phi = \sum_{j=1}^{n} \frac{G_j}{G_\Sigma}\frac{V_{dd}}{V_s}\left(x_j \oplus s_j\right) + \left[\frac{G_{n+1}}{G_\Sigma}\frac{V_{dd}}{V_s} - \phi\right] \tag{67}$$

by comparison to the Boltzmann machine conditional, we can see that the first term implements the model weights (which can be positive or negative given an appropriate setting of the sign bit s_j), and the second term implements a bias.

The static power drawn by this circuit can be written in the form,

$$P^\infty = \frac{C}{\tau_{bias}} V_{dd}^2 (1-\gamma)\gamma \tag{68}$$

where $0 \leq \gamma \leq 1$ is the input-dependent constant,

$$\gamma = \sum_{j=1}^{n+2} \frac{G_j}{G_\Sigma} y_j \tag{69}$$

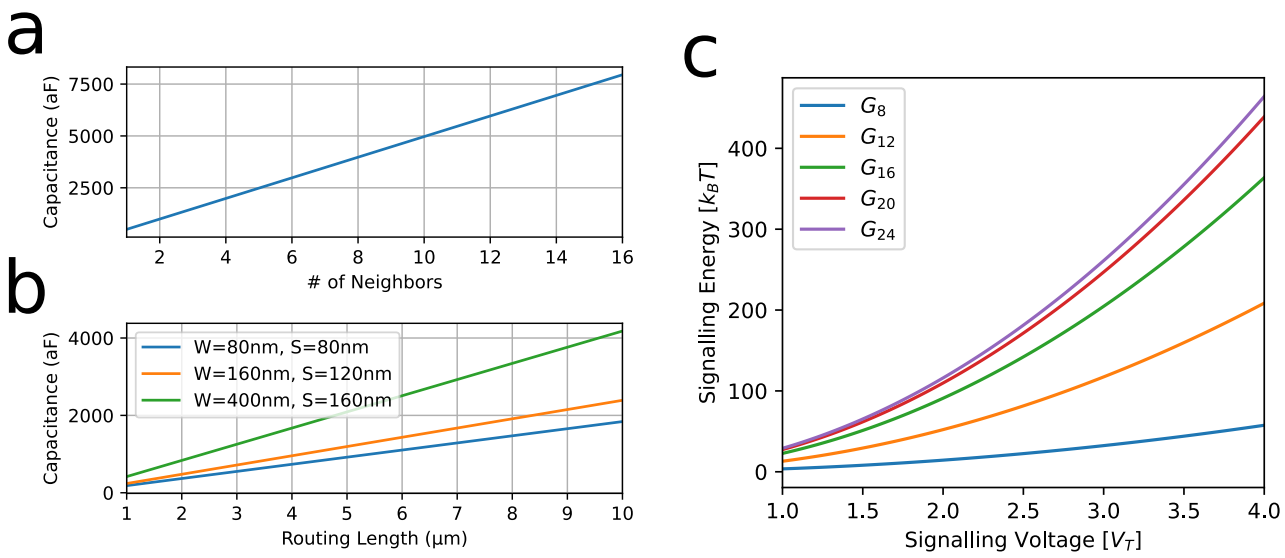

Figure S5. **Parameters for the energy model (a)** The parasitic capacitance associated with the output node of the biasing circuit for various numbers of neighbors. These capacitances were estimated using the PDK and a layout for a real transistor implementation of the biasing circuit. **(b)** The capacitance associated with routing wires of various lengths and geometry in our process, extracted using the PDK. **(c)** The energy required for a cell to signal to all of its neighbors as a function of signaling voltage for various connectivity patterns. This energy was calculated using the routing capacitance data from (b).

This fixed point must be held while the noise generator relaxes, which means that the energetic cost of the biasing circuit is approximately,

$$\begin{aligned} E_{bias} \approx & P^{\infty}\tau_{rng} \\ = & C\frac{\tau_{rng}}{\tau_{bias}}V_{dd}^2(1-\gamma)\gamma \end{aligned} \tag{70}$$

This is maximized for $\gamma = \frac{1}{2}$.

To avoid slowing down the sampling machine, $\frac{\tau_{rng}}{\tau_{bias}} \gg 1$. As such, ignoring the energy spent charging the capacitor $\sim \frac{1}{2}CV_b^2$ will not significantly affect the results, and the approximation made in Eq. (70) should be accurate. The energy consumed by the bias circuit is primarily due to static power dissipation.

2. Local communication

Another significant source of energy consumption is the communication of state information between neighboring cells. In most electronic devices, signals are communicated by charging and discharging wires. Charging a wire requires the energy input,

$$E_{\text{charge}} = \frac{1}{2}C_{\text{wire}}V_{\text{sig}}^2 \tag{71}$$

where C_{wire} is the capacitance associated with the wire, which grows with its length, and V_{sig} is the signaling voltage level.

Given the connectivity patterns shown in table 2, it is possible to estimate the total capacitance C_n associated with the wire connecting a node to all of its neighbors,

$$C_n = 4\eta\ell\sum_i \sqrt{a_i^2 + b_i^2} \tag{72}$$

where $\ell \approx 6\mu m$ is the sampling cell side length, and $\eta \approx 350\text{aF}/\mu m$ is the wire capacitance per unit length in our process, see Fig. 5 (b). a_i and b_i are the x and y components of the i^{th} connection rule, as described in section D.2.

The charging energy Eq. (71) is plotted as a function of signaling voltage for various connectivity patterns in Fig. 5 (b).

3. Global communication

Several systems on the chip require signals to be transmitted from some central location out to the individual sampling cells. This communication involves sending signals over long wires with a large capacitance, which is energetically expensive. Here, the cost of this global communication will be taken into consideration.

a. Clocking

Although it is possible in principle to implement Gibbs sampling completely asynchronously, in practice, it is more efficient to implement standard chromatic Gibbs sampling with a global clock. A global clock requires a signal to be distributed from a central clock circuit to every sampling cell on the chip. This signal distribution is typically accomplished using a clock tree, a branching circuit designed to minimize timing inconsistencies between disparate circuit elements.

To simplify the analysis, we will consider a simple clock distribution scheme in which the clock is distributed by lines that run the entire length of each row in the grid. The total length of the wires used for clock distribution in this scheme is,

$$L_{clock} = N\,L \tag{73}$$

where N is the number of rows in the grid, and L is the length of a row. Given this length, the energetic cost of a clock pulse can be calculated using Eq. (71).

b. Initialization and readout

A sampling program begins by initializing every sampling cell to a specific state and ends by reading out the state of a subset of the cells for use off-chip. Both of these operations require bits to be sent over a long wire of length L from the chip's boundaries to a sampling cell in the bulk.

4. Analysis of a complete sampling program

Given the above analysis of the various subsystems, it is straightforward to construct a model of the energy consumption of a complete denoising model. Running each layer of the denoising model requires initialization of all N nodes, chromatic Gibbs sampling for K iterations, and finally, readout of the N_{data} data nodes,

$$E = T\left(E_{\text{samp}} + E_{\text{init}} + E_{\text{read}}\right) \tag{74}$$

E_{samp} is the cost associated with the sampling iterations for each layer,

$$E_{\text{samp}} = KN\left(E_{\text{rng}} + E_{\text{bias}} + E_{\text{clock}} + E_{\text{nb}}\right) \tag{75}$$

where E_{clock} and E_{nb} are the per-cell costs associated with clock distribution and neighbor communication, respectively.

E_{init} is the cost of initializing all the cells at the beginning of the program,

$$E_{\text{init}} = N\frac{1}{2}\eta L V_{sig}^2 \tag{76}$$

and E_{read} is the cost of reading out the data cells at the end,

$$E_{\text{read}} = N_{\text{data}}\frac{1}{2}\eta L V_{sig}^2 \tag{77}$$

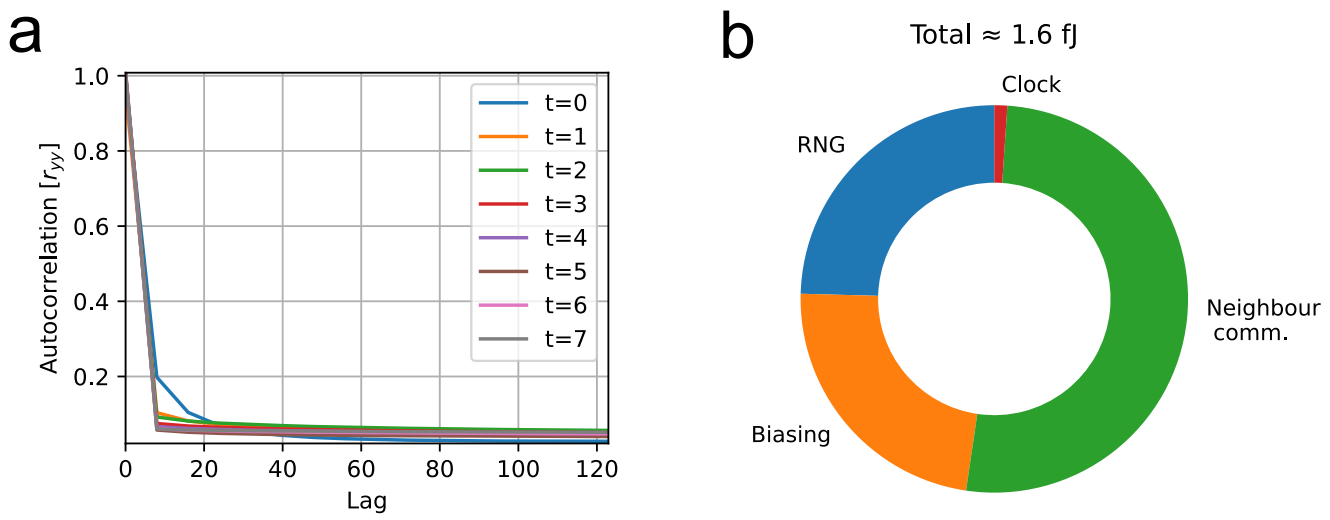

Figure S6. (a) **Autocorrelation curve of a denoising model composed of Boltzmann machines.** Each line represents the autocorrelation of one of the Boltzmann machines that make up a fully trained denoising model.
(b) **Breakdown of the energetic cost of running a sampling cell.** Here, we take $\tau_{rng}/\tau_{bias} = 15$ and $\gamma = 1/2$. We also assume that signaling to neighbors is conducted at a voltage of $4V_T$ (where V_T is the thermal voltage k_BT/e) and the clocking and read/write operations are conducted at a signal level of $5V_T$.

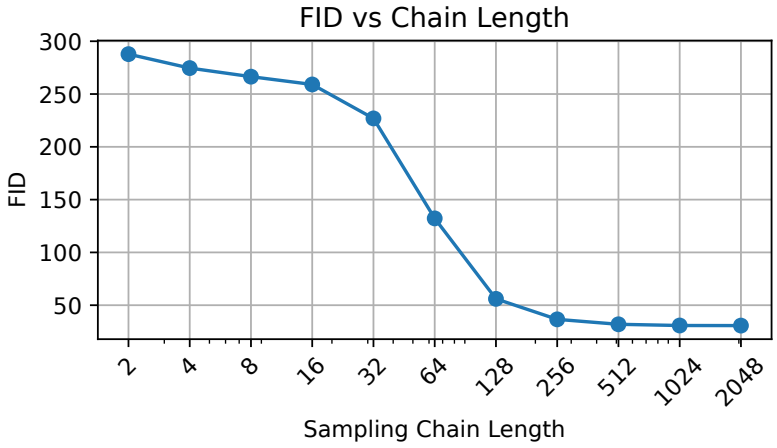

Figure S7. The quality of output images generated by our denoising models stops improving when we sample for more than $K \approx 250$ steps.

This model was used to estimate the energy consumption of the denoising model depicted in Fig. 1 of the article. The mixing behavior for each layer in this denoising model is shown in Fig. 6 (a). All of the layers mix in tens of iterations, with the first layer decaying the most slowly. For the sake of energy calculations, we used $K = 250$ for all layers to be conservative. Fig. 7 shows that for our trained denoising models, sampling for more than $K \approx 250$ steps brings almost no additional benefit, which supports our use of this number for energy calculation. This grid used for each EBM in this model consisted of $N = 4900$ nodes that were connected using a G_{12} pattern. $N_{\text{data}} = 834$ of the nodes were assigned to data, and the rest were latent.

Given realistic choices for the rest of the free parameters of the model, the energetic cost of this denoising model is estimated to be around 1.6 TnJ. This is almost entirely dominated by E_{samp}, with $E_{\text{init}} + E_{\text{read}} \approx 0.01\,T$nJ. A breakdown of the various contributions to E_{samp}, along with more details about the used model parameters, is given in Fig. 6 (b).

This exact procedure was used to estimate the energy consumption of the MEBMs in Fig. 1 in the article. In this case, $T = 1$ and K were estimated from the autocorrelation data for each layer; see section L.

5. Level of realism

The model presented here captures all of the central functional units of a hardware Boltzmann machine sampler. However, the analysis was performed at a high level, and the model almost certainly underestimates the actual energy consumption of a complete device. In practice, when comparing the results of this type of calculation to a detailed analysis of a complete device design, we generally find agreement within an order of magnitude. Given that the gap between conventional methods and our novel hardware architecture is at least several orders of magnitude, this low-resolution analysis is useful, as it supports the claims made in this article without getting into every implementation detail.

Some of the discrepancies between the high-level and detailed model can be attributed to overheads associated with real circuits. A real implementation of the biasing circuit discussed in section E.1 is more complicated than the theoretical model because tunable resistors do not exist. Communications with neighboring cells over long wires require driver circuits, which consume additional energy beyond what is spent charging the line. Despite this, real circuits are bound by the same fundamental physics as the simplified models presented here. As such, the simplified models tend to estimate energy consumption within a factor of two or three of real-life values.

A real device also has additional supporting circuitry compared to our stripped-down model. In the remainder of this section, we will discuss some examples of such supporting circuitry and argue that their contributions to energy consumption at the system level ought not to be significant.

a. Programming the weights and biases

Section E.1 discusses a simple circuit that uses resistors to implement the multiply-accumulate required by the conditional update rule. Key to this is being able to tune the conductance of the resistors to implement specific sets of weights and biases (see Eq. (67)).

Practically, implementing this tunability requires that the model parameters be stored in memory somewhere on the chip. Writing to and maintaining these memories costs energy.

Writing to the memories uses much more energy than maintaining the state. However, if writes are infrequent (program the device once and then run many sampling programs on it before writing again), then the overall cost of the memory is dominated by maintenance. Luckily, most conventional memories are specifically designed to consume as little energy as possible when not being accessed. As such, in practice, the cost of memory maintenance is small compared to the other costs associated with the sampling cells and does not significantly change the outcome shown in Fig. 6.

b. Off-chip communication

External devices have to communicate with our chip for it to be useful. The cost of this communication depends strongly on the tightness of integration between the two systems and is impossible to reason about at an abstract level. As such, the analysis of communication here (as in Section E 3 b) was limited to the cost of getting bits out to the edge of our chip, which is a lower bound on the actual cost.

However, we have found that a more detailed analysis, which includes the cost of communication between two chips mediated by a PCB, does not significantly change the results at the system level. The fundamental reason for this is that sampling programs for complex models run for many iterations before mixing and sending the results back to the outside world. This is reflected in the discrepancy between E_{samp} and $E_{\text{init}} + E_{\text{read}}$ found in section E.4.

c. Supporting circuitry

Any real chip has digital and analog supporting circuitry that provides basic functionality, such as clocking and communication, allowing the rest of the chip to function correctly. The fraction of the energy budget spent on this supporting circuitry generally depends on its size compared to the core computer. Due to the heterogeneity of our architecture, it is possible to share most of the supporting circuitry among many sampling cells, which dramatically reduces the per-cell cost. As such, the energy cost of the supporting circuitry is not significant at the system level.

Supplement F: Energy analysis of GPUs

All experiments shown in Fig. 1 in the article were conducted on NVIDIA A100 GPUs. The empirical estimates of energy were conducted by drawing a batch of samples from the model and measuring the GPU energy consumption and time via Zeus [5]. The theoretical energy estimates were derived by taking the number of model FLOPS (via JAX and PyTorch's internal estimators) and plugging them into the NVIDIA GPU specifications (19.5 TFLOPS for Float32 and 400W). The empirical measurements are compared to theoretical estimates for the VAE in Table 3, and the empirical measurements show good alignment with the theoretical.

FID	Empirical Efficiency	Theoretical Efficiency
30.5	6.1×10^{-5}	2.3×10^{-5}
27.4	1.5×10^{-4}	0.4×10^{-4}
17.9	2.5×10^{-3}	1.7×10^{-3}

Table S3. Comparing theoretical vs empirical energy consumption for a VAE on a GPU. Energy efficiencies are reported in units of joules per sample.

The models were derived from available implementations and are based on small versions of ResNet [6] and UNet [7] style architectures. Their FID performance is consistent with published literature values [8–10]. The goal is not to achieve state of the art performance, but to represent the relative scales of energy consumption of the algorithms. For a direct simulation of the Ising models on a GPU, theoretical efficiency on the order of 10^{-4} joules per sample comparable in performance and efficiency to the small VAE.

The reader may be surprised to see that the diffusion model is substantially less energy-efficient than the VAE given the relative dominance in image generation. However, two points should be kept in mind. First, while VAE remains a semi-competitive model for these smaller datasets, this quickly breaks down. On larger datasets, a FID performance gap usually exists between diffusion models and VAEs. Second, these diffusion models (based on the original DDPM [2]) have performance that can depend on the number of diffusion time steps. So, not only is the UNet model often larger than a VAE decoder, but it also must be run dozens to thousands of times in order to generate a single sample (thus resulting in multiple orders of magnitude more energy required). Modern improvements, such as distillation [11], may move the diffusion model energy efficiency closer to the VAE's.

Supplement G: Autocorrelation and mixing time

We monitor the mixing of our Gibbs samplers via the autocorrelation of a low-dimensional projection of the Markov chain state.

Let $\{x[j]\}_{j\geq 0}$ be a discrete-time Markov chain on a finite state space $\{1, \dots, d\}$, with time-homogeneous transition kernel $P = (p_{xy})_{1\leq x,y\leq d}$ given by

$$p_{xy} = \mathbb{P}(x[j+1] = y \mid x[j] = x).$$

In our setting, $x[j]$ is the state of the Boltzmann machine at Gibbs iteration j.

We fix a function (projection) $f : \{1, \dots, d\} \to \mathbb{R}$ and define the scalar observable

$$y[j] = f(x[j]).$$

We then define the mean

$$\mu \equiv \mathbb{E}\big[y[j]\big] \tag{78}$$

and the (normalized) autocorrelation function

$$r_{yy}[k] = \frac{\mathbb{E}\big[(y[j]-\mu)\,(y[j+k]-\mu)\big]}{\mathbb{E}\big[(y[j]-\mu)^2\big]}, \qquad k \in \mathbb{N}. \tag{79}$$

Here $\mathbb{E}[\cdot]$ denotes expectation with respect to the joint law of the Markov chain. In practice, we approximate this expectation by averaging over time and across multiple independent Gibbs sampling chains.

For all experiments in this article, we take f to be the encoder network used in the FID computation, applied to the

visible states of the Boltzmann machine. This choice is largely arbitrary: we found that much simpler embeddings, such as random linear projections $y[j] = Ax[j]$, behave similarly well for the purposes of autocorrelation-based mixing diagnostics.

Spectral decomposition and decay of autocorrelation

We now briefly recall how the decay rate of autocorrelation is tied to the spectral properties of the transition kernel P, and hence to the mixing time of the Markov chain. Proofs of the statements below can be found in standard references such as [12].

Assume the Markov chain $\{x[j]\}_{j\geq 0}$ is

- **irreducible**, i.e. any state can be reached from any other in a finite number of steps, and
- **aperiodic**, i.e. for any $x \in \{1, \dots, d\}$ there exists $T \in \mathbb{N}$ such that for all $t \geq T$, $\mathbb{P}(x[t] = x \mid x[0] = x) > 0$.

Since the chain is finite and irreducible, there exists a unique stationary distribution π satisfying

$$\pi = \pi P.$$

Furthermore, aperiodicity implies that, for any initial distribution ψ_0 of $x[0]$, the law ψ_t of $x[t]$ converges to π as $t \to \infty$:

$$\psi_t = \psi_0 P^t \;\longrightarrow\; \pi \qquad \text{as } t \to \infty.$$

Assume further that P is diagonalizable with real eigenvalues (which is always true if the chain is reversible, like in the case of Gibbs sampling), and write its eigendecomposition as

$$P = U^{-1} \Sigma U,$$

where Σ is diagonal, and its entries are ordered

$$1 = \sigma_1 > \sigma_2 \geq \cdots \geq \sigma_d \geq 0.$$

We denote by $U(i, x)$ the entry in the ith row and xth column of U (and similarly for U^{-1}). The first left eigenvector of P (the first row of U) is the stationary distribution,

$$\pi = \pi P = U(1, \cdot),$$

and the first right eigenvector is a column of ones,

$$U^{-1}(\cdot, 1) = \mathbf{1}.$$

Let $f : \{1, \dots, d\} \to \mathbb{R}$ be the same function as above, and write

$$\mu_0^f \equiv \mathbb{E}_{Y\sim\psi_0}\big[f(Y)\big] = \mathbb{E}\big[f(x[0])\big], \qquad \mu_\infty^f \equiv \mathbb{E}_{Y\sim\pi}\big[f(Y)\big] = \lim_{t\to\infty} \mathbb{E}\big[f(x[t])\big], \tag{80}$$

where ψ_0 is the initial distribution of $x[0]$.

Let δ_k denote the column vector with a 1 in position k and 0 elsewhere. Using the eigendecomposition of P, the transition probabilities can be written as

$$(\delta_{x_0}^T P^t)(x) = \sum_{j=1}^{d} U^{-1}(x_0, j)\, \sigma_j^t\, U(j, x).$$

Thus,

$$\begin{aligned}
\mathbb{E}\big[f(x[0])f(x[t])\big] &= \sum_{x_0=1}^{d} f(x_0)\,\mathbb{P}\big[x[0]=x_0\big]\,\mathbb{E}\big[f(x[t]) \mid x[0]=x_0\big] \\
&= \sum_{x_0=1}^{d}\sum_{x=1}^{d} f(x_0)f(x)\,\psi_0(x_0)\,(\delta_{x_0}^T P^t)(x) \\
&= \sum_{x_0=1}^{d}\sum_{x=1}^{d} f(x_0)f(x)\,\psi_0(x_0)\sum_{j=1}^{d} U^{-1}(x_0,j)\,\sigma_j^t\,U(j,x) \\
&= \left(\sum_{x_0=1}^{d}\sum_{x=1}^{d} f(x_0)f(x)\,\psi_0(x_0)\,U^{-1}(x_0,1)\,\sigma_1^t\,U(1,x)\right) \\
&\quad + \sum_{j=2}^{d}\sigma_j^t \sum_{x_0=1}^{d}\sum_{x=1}^{d} f(x_0)f(x)\,\psi_0(x_0)\,U^{-1}(x_0,j)\,U(j,x).
\end{aligned} \tag{81}$$

Using $U^{-1}(\cdot,1)=\mathbf{1}$ and $U(1,\cdot)=\pi$, the first term simplifies to

$$\sum_{x_0=1}^{d}\sum_{x=1}^{d} f(x_0)f(x)\,\psi_0(x_0)\,\pi(x) = \mu_0^f\mu_\infty^f.$$

We can therefore write

$$\mathbb{E}\big[f(x[0])f(x[t])\big] = \mu_0^f\mu_\infty^f + \sum_{j=2}^{d}\sigma_j^t c_j, \tag{82}$$

where the coefficients c_j are constants (depending on f and ψ_0 but not on t). For large t, the terms with smaller eigenvalues become negligible compared to the contribution from σ_2, so the covariance

$$\mathbb{E}\big[f(x[0])f(x[t])\big] - \mu_0^f\mu_\infty^f$$

decays asymptotically like σ_2^t. In particular, under stationarity ($\psi_0=\pi$ so that $\mu_0^f=\mu_\infty^f=\mu$), the autocorrelation $r_{yy}[t]$ defined in (79) decays approximately as

$$r_{yy}[t] \approx C\,\sigma_2^t$$

for some constant C depending on f.

From eigenvalues to mixing time

The spectral quantity σ_2 is directly related to the mixing time of the Markov chain. Recall that the mixing time is defined by

$$\tau(\varepsilon) = \min\left\{t \geq 0 : \max_{\psi_0} \|\psi_0 P^t - \pi\|_{\mathrm{TV}} \leq \varepsilon\right\},$$

where

$$\|\mu-\nu\|_{\mathrm{TV}} = \frac{1}{2}\sum_{x=1}^{d}|\mu(x)-\nu(x)|$$

denotes the total variation distance and $\varepsilon>0$ is a prescribed tolerance.

Using the eigendecomposition $P = U^{-1}\Sigma U$, we can write

$$\psi_0 P^t(x) = \sum_{j=1}^{d} \big(\psi_0 \cdot U^{-1}(\cdot, j)\big)\, \sigma_j^t\, U(j, x),$$

where ψ_0 is treated as a row vector and $\psi_0 \cdot U^{-1}(\cdot, j)$ denotes the scalar product with the jth column of U^{-1}. Using $\psi_0 \cdot U^{-1}(\cdot, 1) = 1$ and $U(1, \cdot) = \pi$, we obtain

$$\begin{aligned}
\|\psi_0 P^t - \pi\|_{\mathrm{TV}} &= \frac{1}{2}\sum_{x=1}^{d} \left| \pi(x) - \sum_{j=1}^{d} \big(\psi_0 \cdot U^{-1}(\cdot, j)\big)\, \sigma_j^t\, U(j, x) \right| \\
&= \frac{1}{2}\sum_{x=1}^{d} \left| \pi(x) - \pi(x) + \sum_{j=2}^{d} \big(\psi_0 \cdot U^{-1}(\cdot, j)\big)\, \sigma_j^t\, U(j, x) \right| \\
&\leq \frac{1}{2}\sum_{x=1}^{d}\sum_{j=2}^{d} \big|\psi_0 \cdot U^{-1}(\cdot, j)\big|\, |\sigma_j^t|\, |U(j, x)| \\
&= \sum_{j=2}^{d} \sigma_j^t\, a_j \;\leq\; \sigma_2^t \sum_{j=2}^{d} a_j,
\end{aligned}$$

where $a_j \geq 0$ are constants that depend on ψ_0 and on the eigenvectors of P. Therefore, the total variation distance decays at least as fast as σ_2^t, and we obtain the upper bound

$$\tau(\varepsilon) \;\leq\; \frac{\log(\varepsilon) - \log\Big(\sum_{j=2}^{d} a_j\Big)}{\log(\sigma_2)}.$$

Since $0 \leq \sigma_2 < 1$, the denominator $\log(\sigma_2)$ is negative, and the right-hand side is positive. The smaller the value of σ_2, the faster the Markov chain mixes, and the more rapidly the autocorrelation $r_{yy}[k]$ decays.

In summary, by empirically estimating the long-lag decay of the autocorrelation function $r_{yy}[k]$ for some informative observable $f(x)$, we effectively probe the second-largest eigenvalue σ_2 of the transition kernel and thus obtain a proxy for the mixing time of the Gibbs sampler underlying our DTM.

Supplement H: Total correlation penalty

In the main text (see Eq. 17), we explain how we utilize a total correlation penalty to encourage the latent variable EBMs employed in our model to mix rapidly. Here, we will discuss a few details of this regularizer and the method we use to control its strength adaptively.

1. Gradients of the total correlation penalty

The total correlation penalty is a convenient choice in this context because its gradients can be computed using the same samples used to estimate the gradient of the usual loss used in training, $\nabla_\theta \mathcal{L}_{DN}$. Namely, treating the factorized distribution as a constant with respect to the gradient,

$$\nabla_\theta \mathcal{L}_t^{TC} = \mathbb{E}_{Q(x^{t-1})}\left[\mathbb{E}_{d(s^{t-1}|x^t)}\left[\nabla_\theta \mathcal{E}_{t-1}^\theta\right] - \mathbb{E}_{P_\theta(s^{t-1}|x^t))}\left[\nabla_\theta \mathcal{E}_{t-1}^\theta\right]\right] \tag{83}$$

where,

$$d(s^{t-1}|x^t) = \prod_{i=1}^{M} P_\theta(s_i^{t-1}|x^t) \tag{84}$$

The second term in Eq. (83) also appears in the estimator for $\nabla_\theta \mathcal{L}_{DN}$. The first term can be simplified when $\mathcal{E}_{t-1}^\theta$

has particular symmetries. For example, if $\mathcal{E}^{\theta}_{t-1}$ is a Boltzmann machine energy function (see main text Eq. 10),

$$\mathbb{E}_{d(s^{t-1}|x^t)}\left[\frac{d}{dh_i}\mathcal{E}^{\theta}_{t-1}\right] = -\beta\,\mathbb{E}_{P_\theta(s_i|x_t)}\left[s_i\right] \tag{85}$$

$$\mathbb{E}_{d(s^{t-1}|x^t)}\left[\frac{d}{dJ_{ij}}\mathcal{E}^{\theta}_{t-1}\right] = -\beta\,\mathbb{E}_{P_\theta(s_i|x_t)}\left[s_i\right]\;\mathbb{E}_{P_\theta(s_j|x_t)}\left[s_j\right] \tag{86}$$

Each of these terms is easy to compute given the samples used to estimate $\nabla_\theta \mathcal{L}_{DN}$.

2. Adaptive Correlation Penalty

The optimal strength of the correlation penalty λ_t may vary depending on the specific denoising step t (models for less noisy data near $t = 0$ may require stronger regularization) and may even change during training for a single-step model. Manually tuning λ_t for each of the step-models would be prohibitively expensive.

To address this, we employ an Adaptive Correlation Penalty (ACP) scheme that dynamically adjusts λ_t based on an estimate of the model's current mixing time. We use the autocorrelation of the Gibbs sampling chain, r^t_{yy}, as a proxy for mixing, as described in Section G and the main text, Eq. 18.

Our ACP algorithm monitors the autocorrelation at a lag K equal to the number of Gibbs steps used in the estimation of $\nabla_\theta \mathcal{L}_{DN}$. The goal is to adjust γ_{CP} to keep this autocorrelation below a predefined target threshold ε_{ACP}.

A simple layerwise procedure is used for this control. The inputs to the algorithm are the initial values of λ_t, a target autocorrelation threshold ε_{ACP} (e.g., 0.03), an update factor δ_{ACP} (e.g., 0.2) and a lower limit $\lambda^{\min}_t$ (e.g., 0.0001).

At the end of each training epoch m:

1. Estimate the current autocorrelation $a^t_m = r^t_{yy}[K]$. This estimate can be done by running a longer Gibbs chain periodically and calculating the empirical autocorrelation from the samples.
2. Set $\lambda'_t = max(\lambda^{\min}_t, \lambda^{(m)}_t)$ to avoid getting stuck at 0.
3. Update λ_t for the next epoch $(m + 1)$ based on a^t_m and the previous value a^t_{m-1} (if $m > 0$):
   - If $a^t_m < \varepsilon_{\text{ACP}}$: The chain mixes sufficiently fast; reduce the penalty slightly.
   $$\lambda^{(m+1)}_t \leftarrow (1 - \delta_{\text{ACP}})\lambda'_t$$
   - Else if $a^t_m \geq \varepsilon_{\text{ACP}}$ and $a^t_m \leq a^t_{m-1}$ (or $m = 0$): Mixing is slow but not worsening (or baseline); keep the penalty strength.
   $$\lambda^{(m+1)}_t \leftarrow \lambda'_t$$
   - Else ($a^t_m > \varepsilon_{\text{ACP}}$ and $a^t_m > a^t_{m-1}$): Mixing is slow and worsening; increase the penalty.
   $$\lambda^{(m+1)}_t \leftarrow (1 + \delta_{\text{ACP}})\lambda'_t$$
4. If the proposed value $\lambda^{(m+1)}_t < \lambda^{\min}_t$, then set $\lambda^{(m+1)}_t \leftarrow 0$.

Our experiments indicate that this simple feedback mechanism works effectively. While λ_t and the autocorrelation a^t_m might exhibit some damped oscillations for several epochs before stabilizing this automated procedure is vastly more efficient than performing manual hyperparameter searches for λ_t for each of the T models.

Training is relatively insensitive to the exact choice of ε_{ACP} within a reasonable range (e.g., $[0.02, 0.1]$) and δ_{ACP} (e.g., $[0.1, 0.3]$). Assuming that over the course of training the λ_t parameter settles around some value λ^*_t, one should aim for the lower bound parameter $\lambda^{\min}_t$ to be smaller than $\frac{1}{2}\lambda^*_t$, while making sure that the ramp-up time $\frac{\log(\lambda^*_t)-\log(\lambda^{\min}_t)}{\log(1+\delta_{\text{ACP}})}$ remains small. Settings of $\lambda^{\min}_t$ in the range $[0.001, 0.00001]$ all produced largely the same result, the only difference being that values on the lower end of that range led to a larger amplitude in oscillations of λ_t and a^t_m, but training eventually settled for all values. An example of some ACP dynamics is shown in Fig. 8:

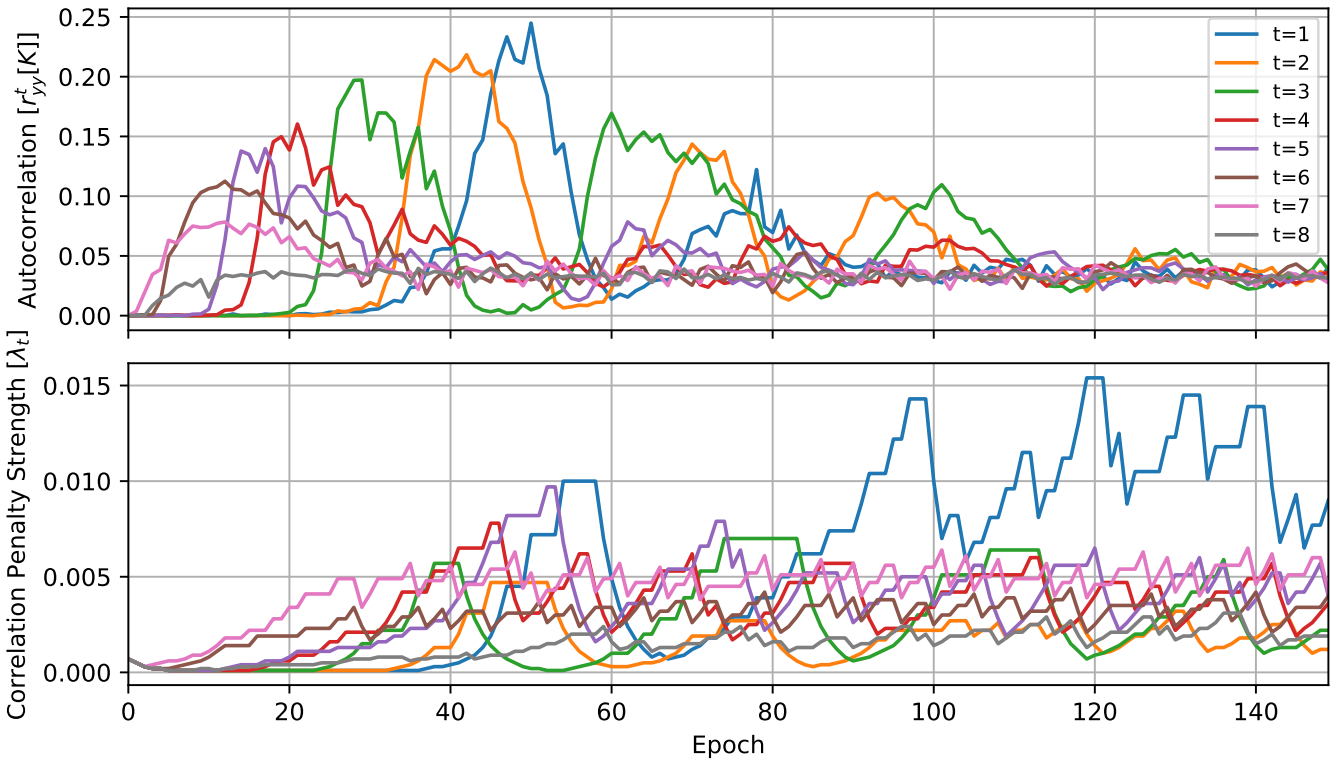

Figure S8. **The adaptive correlation penalty** The dynamics of r^t_{yy} and λ_t over a training run. Large values of r^t_{yy} lead to increasingly large values of λ_t, which cause r^t_{yy} to decrease. The system reaches a stable configuration by the end of training.

Training on Fashion-MNIST with the typical experimental setup, we observed nearly the same performance (a FID of 28 ± 1) for all choices of ε_{ACP}, δ_{ACP} and $\lambda_t^{\min}$ in the ranges written above, so long as we trained for at least 100 epochs (with specific settings the training took longer to converge).

Supplement I: Embedding integers into Boltzmann machines

In some of our experiments, we needed to embed continuous data into binary variables. We chose to do this by representing a k-state categorical variable X_i using the sum k binary variables Z_i^k,

$$X_i = \sum_{k=1}^{K_i} Z_i^{(k)} \tag{87}$$

where $Z_i^{(k)} \in \{0, 1\}$. These binary variables can be trivially converted into spin variables that are $\{-1, 1\}$ valued using a linear change of variables.

Energy functions that involve quadratic interactions between these categorical variables can be reduced to Boltzmann machines with local patches of all-to-all connectivity. For example, consider the energy function,

$$E(x; \theta) = -\sum_{i \neq j} w_{ij} X_i X_j - \sum_{i=1}^{d} b_i X_i \tag{88}$$

inserting Eq. (87), we can rewrite this in terms of quadratic interactions between the underlying spins $Z_i^{(k)}$,

$$E(z; \theta) = -\sum_{i \neq j} w_{ij} \left(\sum_{k=1}^{K_i} Z_i^{(k)} \right) \left(\sum_{l=1}^{K_j} Z_j^{(l)} \right) - \sum_i b_i \left(\sum_{k=1}^{K_i} Z_i^{(k)} \right) \tag{89}$$

which is a standard Boltzmann machine energy function that can be run on our hardware, just like any other.

Supplement J: Deterministic embeddings for DTMs

In the section on Hybrid Thermodynamic-Deterministic Machine Learning (HTDML) in the paper, we mention hybrid thermodynamic models, the purpose of which is to combine the flexibility of conventional neural networks (NNs) with the efficiency of probabilistic computers. For example, in the context of image generation, a small convolutional neural network can be used to map color images into a format compatible with a binary DTM. To properly take advantage of the DTM's energy efficiency, the conventional model should be at least 2 or 3 orders of magnitude smaller (e.g., in terms of parameter count or number of operations per sample) than the DTM.

There are various options for the type of conventional model one can use for the embedding, e.g., invertible models such as GLOW [13] or Normalizing Flows [14], as well as simpler solutions, such as an Autoencoder.

For our proof-of-concept for hybrid models, we used a combination of an Autoencoder and a GAN [15].

- First, we train a convolutional Autoencoder (encoder plus decoder) that maps images into a binary latent space (achieved through a combination of a sigmoid activation, a binarization penalty, and a straight-through gradient).

- Second, we train a DTM on latent embeddings of the training images. At inference time, the samples generated by the DTM are passed through the decoder to produce images.

- Thirdly, we use a GAN-like approach to fine-tune the decoder to utilize the outputs of the Boltzmann machine maximally. Specifically, the Boltzmann machine outputs are used as the noise source, which is fed into the decoder (now taking the role of the generator in the GAN architecture), and finally, a critic is trained to guide the decoder towards generating higher-quality images.

Our hybrid model achieved a FID score of ~ 60 on CIFAR10. Our DTM had 8 million parameters, the decoder had 65k, and the encoder and critic were both below 500k parameters. At inference time, only the DTM and the decoder are used. To achieve a similar performance with a conventional GAN, the decoder/generator requires about 500000 parameters.

Supplement K: An all-transistor RNG

Our RNG is a digitizing comparator fed by a source of Gaussian noise. The noise source is implemented using the circuitry and principles described in [16]. The comparator is a standard design that operates in subthreshold to minimize energy consumption. The mean of the Gaussian noise is shifted before it is sent into the comparator to implement the bias control. A schematic of our RNG is shown in Fig. 9 (a).

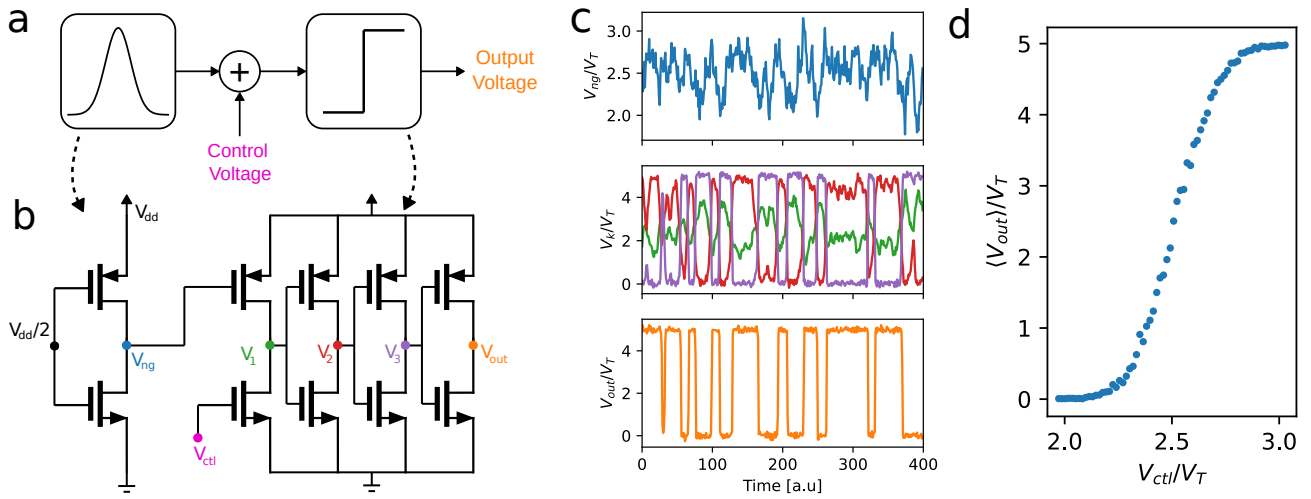

Figure S9. **An example circuit implementation of the RNG**. **(a)** A high-level schematic of our RNG design. **(b)** A simple circuit implementation of the architecture given in (a). A single CMOS inverter produces gaussian noise, which is digitized by the four-inverter chain. **(c)** Example output waveforms of the circuit shown in (b). The inverter chain gradually digitizes the noise produced by the first stage. **(d)** The sigmoidal bias curve of the circuit given in (b).

Fig. 9 (b) shows a simplified version of the circuit that implements our RNG: the noise source is implemented by a CMOS inverter as described in [16], and the comparator is implemented by four more inverters that approximately digitize the noise. Fig. 9 (c) shows how the initial noisy voltage V_{ng} is gradually digitized by the inverter chain, producing an output voltage waveform that randomly wanders between 0 and 1.

The bias of the example circuit is a sigmoidal function of a single control voltage, V_{ctl}, as shown in Fig. 9 (d). Connecting this terminal to the output of a biasing circuit (as described in Section E.1) allows for the implementation of the Boltzmann machine conditional update rule.

The sigmoidal biasing curve is a fairly generic feature of many possible subthreshold RNG designs. The root cause of this is the exponential current-voltage relations that govern subthreshold devices, which result from the fact that charge transport is primarily thermally activated in subthreshold transistors. This is probably true of a wide range of nanoscopic devices; the Bolzmann machine conditional update is "natural" to implement using analog probabilistic hardware.

Significant additions must be made to this simple example to circuit to make it robust to process variation and drift (which is required for it to work reliably in real life). This additional circuitry raises the power consumptuion of a real RNG circuit compared to this ideal example: the circuit shown in Fig. 9 (b) consumes only ~ 75 aJ per random bit (assuming each node has an 800aF capacitance to ground), as compared to the ~ 400 aJ per random bit found for the design used in the main text.

Our RNG was part of the same test chip used to carry out the experiments in [16]. The output of the RNG was fed into an amplification chain that buffered it and allowed its signal to be observed using an external oscilloscope. Fig. 10 (c) shows an image of our packaged test chip, along with a view of our RNG through a 100× microscope objective.

Supplement L: MEBM experiments

Our experiments on MEBMs were conducted in the typical way [17]. We employed the same Boltzmann machine architecture as we typically use for the DTM layers, specifically $L = 70$ with G_{12} connectivity. Random nodes were chosen to represent the data, and the rest were left as latent variables (as discussed in section D).

Generating the data presented in Figs. 1 and 2 in the main text required controlling the mixing time of a trained Boltzmann machine. To achieve this, we added a fixed correlation penalty (Eq. 17 in the main text) and varied the strength to control the allowed complexity of the energy landscape.

Fig. 11 (a) shows an example of the raw autocorrelation curves produced by sampling from Boltzmann machines trained with different correlation penalty strengths. The slowest exponential decay rate (σ_2) could be estimated for most of the curves by fitting a line to the natural log of the autocorrelation curve at long times, see Fig. 11 (b) The two curves with the smallest correlation penalty did not reduce to simple exponential decay during the measured lag values, which means the decay rate was too long to be extracted from our data.

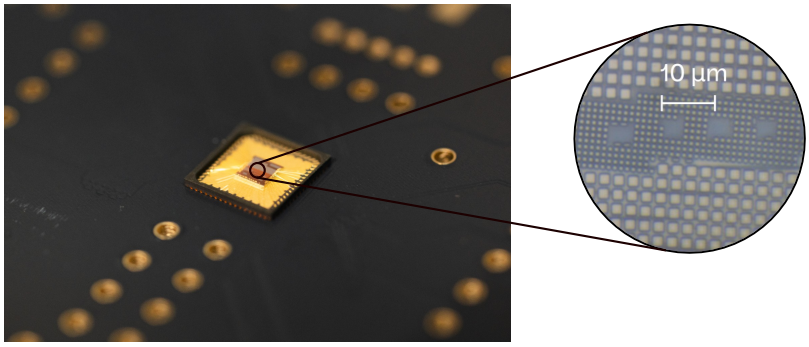

Figure S10. **Our RNG** An image of our packaged test chip (with the top of the package removed) assembled onto a PCB. We also show an optical microscope image of several RNG circuits on our test chip. Each circuit occupies an approximately $3 \times 3\mu m$ area on the chip.

The exponential decay rates extracted from Fig. 11 were used as the mixing times in Fig. 2 in the article. Calling this a "mixing time" is a slight abuse of nomenclature. However, we did not think it made enough of a difference to the article's message to disambiguate (since it is an upper bound on the mixing time, as discussed in Section G).

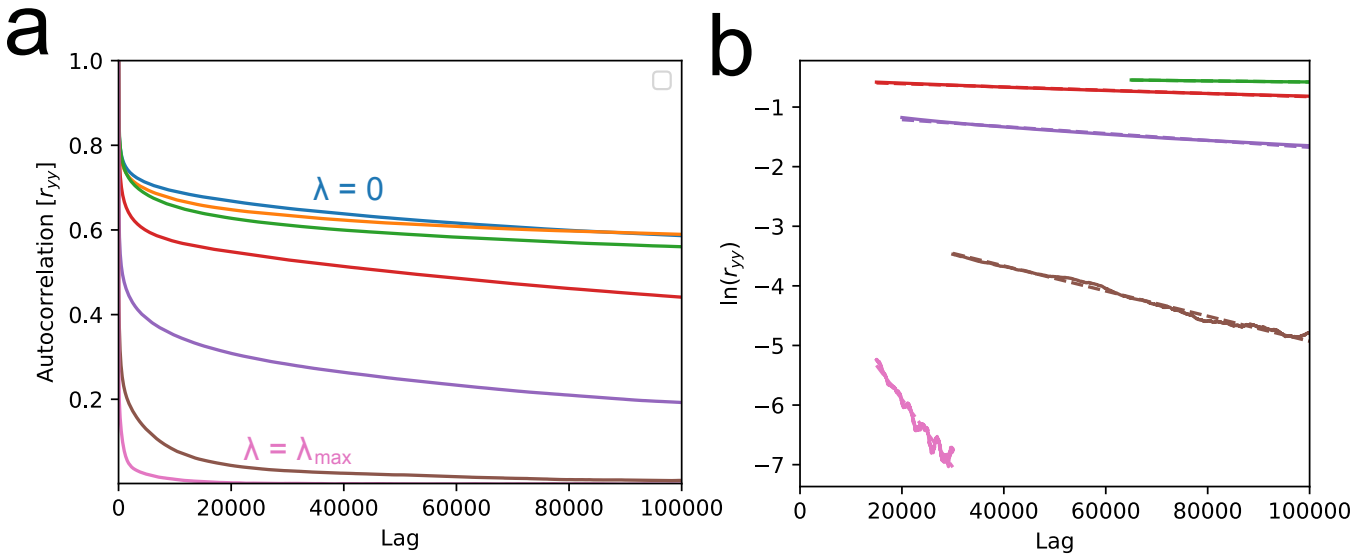

Figure S11. **Boltzmann machine autocorrelation curves (a)** The raw autocorrelation data associated with Boltzmann machines trained using different values of the parameter λ. **(b)** The log of the long-time autocorrelation for some of the curves shown in (a). All curves, except for the blue and orange ones, eventually became linear.

Supplement M: Sampling time vs Performance Tradeoff

The total time it takes to sample from a DTM is $O(KT)$, where T is the number of denoising steps comprising the DTM and K. While one might use a different number of Gibbs steps at training time and inference time, it should be noted that for DTMs trained with adaptive correlation penalty (ACP), $K_{\text{inference}}$ should be similar or only slightly higher to K_{train}. This is substantially different from the usual good practice for un-penalised EBMs. The main reason for this distinction is ACP. In particular, when training an EBM (or DTM) with ACP, the closed-loop control used in ACP will make sure that the mixing time K_{mix} of the EBM is approximately equal to K_{train}. It is always the case that using $K_{\text{inference}} >> K_{\text{mix}}$ brings no benefits compared to $K_{\text{inference}} \approx K_{\text{mix}}$ and therefore when sampling from an ACP-regularised model, using $K_{\text{inference}} \approx K_{\text{train}}$. On the other hand, for usual EBMs trained without ACP (or a similar penalty), the mixing time of the EBM can greatly exceed K_{train}. Due to the mixing-expressivity tradeoff, this at first means that the performance of this un-penalised EBM can exceed that of an ACP EBM, as long as $K_{\text{inference}} \geq K_{\text{mix}} >> K_{\text{train}}$. However, if we then keep training with $K_{\text{train}} << K_{\text{mix}}$, the performance of the EBM

starts to degrade substantially, as shown in fig. 13. One may attempt to circumvent this by treating the EBM as a non-equilibrium model as explored in [18], but that introduces significant new complexity and is mathematically not well understood.

That being said, why does increasing K_{train} improve the performance of an ACP-regularised DTM or MEBM? ACP works by periodically checking whether $K_{\text{mix}} > K_{\text{train}}$. If yes, then it penalizes the model more, reducing K_{mix}, but potentially hurting the expressivity of the model. If not, then it reduces the penalty coefficient, allowing both K_{mix} and model expressivity to increase. Therefore, using a larger K_{train} lets ACP be more lenient, resulting in a more expressive model.

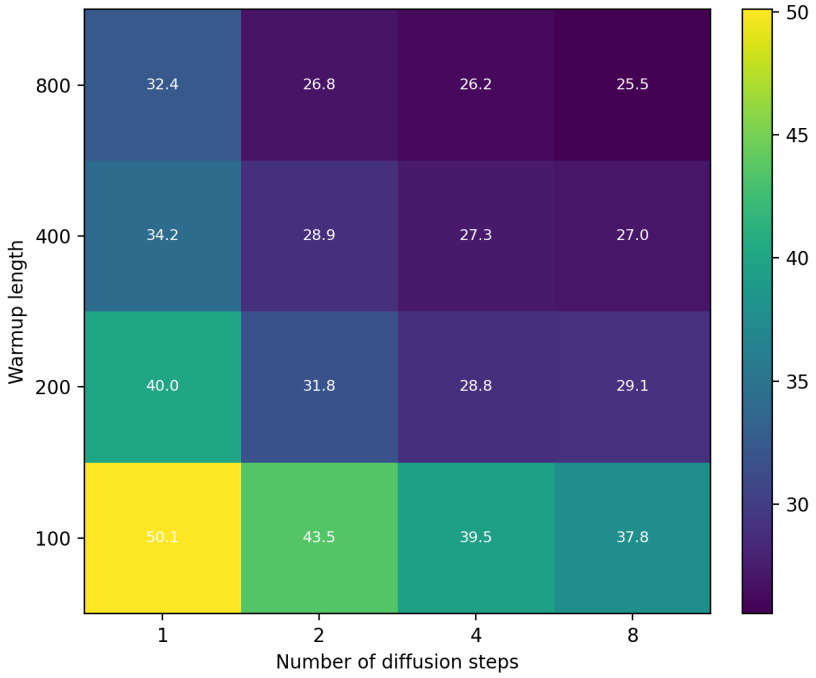

Figure S12. The FID scores of different DTMs trained on binarised FashionMNIST. On the horizontal axis we vary the number of denoising steps T and on the vertical axis we vary the number of Gibbs steps during training K_{train}. Each diagonal (running from top-left to bottom-right) represents a constant amount of energy expenditure. For this plot, the number of Gibbs steps at inference was twice the number used during training. The reported FID values were averaged over three runs with different random seeds.

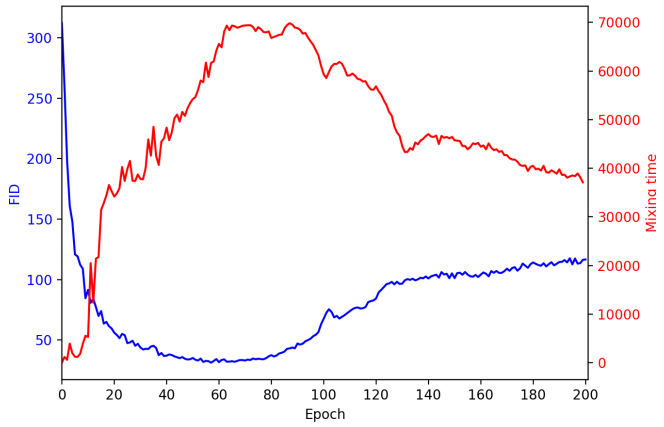

Figure S13. FID and mixing time of an MEBM (same topology as most other experiments in this article) throughout training. As the MEBM trains and becomes more expressive (FID goes down) the mixing time increases. Around epoch 20 the model starts freezing (the mixing time becomes too long), which causes the CD gradients to become inaccurate and soon after the performance of the model starts to degrade. The mixing time subsequently also decreases somewhat, although this is likely an artefact of mode collapse rather than a sign of relaxation of the energy barriers in the model.

Supplement N: CIFAR-10 Images

Figure S14. Images generated by the hybrid model trained on CIFAR-10 as described in the section on Hybrid Thermodynamic-Deterministic Machine Learning in the main text of this paper and in Supplement J. Images come in groups of 8, where each group shows the progression of the image throughout the denoising process. The denoising process operates inside a binary latent space, which is then converted into images by a small neural-network-based decoder.

[1] Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N. & Ganguli, S. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning*, 2256–2265 (pmlr, 2015).

[2] Ho, J., Jain, A. & Abbeel, P. Denoising diffusion probabilistic models. *Advances in neural information processing systems* **33**, 6840–6851 (2020).

[3] Lou, A., Meng, C. & Ermon, S. Discrete diffusion modeling by estimating the ratios of the data distribution (2024). URL https://arxiv.org/abs/2310.16834. 2310.16834.

[4] Murphy, K. P. *Probabilistic Machine Learning: Advanced Topics* (MIT Press, 2023). URL http://probml.github.io/book2.

[5] You, J., Chung, J.-W. & Chowdhury, M. Zeus: Understanding and optimizing {GPU} energy consumption of {DNN} training. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 119–139 (2023).

[6] He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).

[7] Ronneberger, O., Fischer, P. & Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, 234–241 (Springer, 2015).

[8] Dai, B. & Wipf, D. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations* (2019). URL https://openreview.net/forum?id=B1e0X3C9tQ.

[9] Chadebec, C., Vincent, L. & Allassonniere, S. Pythae: Unifying generative autoencoders in python-a benchmarking use case. *Advances in Neural Information Processing Systems* **35**, 21575–21589 (2022).

[10] Ostheimer, P., Nagda, M., Kloft, M. & Fellenz, S. Sparse data generation using diffusion models. *arXiv preprint arXiv:2502.02448* (2025).

[11] Liu, X., Zhang, X., Ma, J., Peng, J. *et al.* Instaflow: One step is enough for high-quality diffusion-based text-to-image generation. In *The Twelfth International Conference on Learning Representations* (2023).

[12] Levin, D., Peres, Y. & Wilmer, E. *Markov Chains and Mixing Times* (American Mathematical Soc., 2009). URL https://books.google.ca/books?id=6Cg5Nq5sSv4C.

[13] Kingma, D. P. & Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In Bengio, S. *et al.* (eds.) *Advances in Neural Information Processing Systems*, vol. 31 (Curran Associates, Inc., 2018). URL https://proceedings.neurips.cc/paper_files/paper/2018/file/d139db6a236200b21cc7f752979132d0-Paper.pdf.

[14] Papamakarios, G., Nalisnick, E., Rezende, D. J., Mohamed, S. & Lakshminarayanan, B. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research* **22**, 1–64 (2021). URL http://jmlr.org/papers/v22/19-1028.html.

[15] Goodfellow, I. J. *et al.* Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. & Weinberger, K. (eds.) *Advances in Neural Information Processing Systems*, vol. 27 (Curran Associates, Inc., 2014). URL https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf.

[16] Freitas, N. *et al.* *Taming non-equilibrium thermal fluctuations in subthreshold CMOS circuits. Phys. Rev. App.* (2025). Accepted.

[17] Sajeeb, M. *et al.* Scalable connectivity for ising machines: Dense to sparse. *arXiv preprint arXiv:2503.01177* (2025).

[18] Decelle, A., Furtlehner, C. & Seoane, B. Equilibrium and non-equilibrium regimes in the learning of restricted boltzmann machines*. *Journal of Statistical Mechanics: Theory and Experiment* **2022**, 114009 (2022). URL http://dx.doi.org/10.1088/1742-5468/ac98a7.