Energy-Aware Benchmarking: A Case Study on Sampling Methods

Anna Kazachkova*, **Johann Ukrow***, **Sven Köhler***, **Nicolas Alder**,
Rainer Schlosser, **Ralf Herbrich**
Hasso Plattner Institute
{anna.kazachkova,johann.ukrow,sven.koehler,nicolas.alder,
rainer.schlosser,ralf.herbrich}@hpi.de

Abstract

Energy efficiency is increasingly recognized as an important dimension in machine learning research, yet systematic approaches to measuring and optimizing energy use remain underdeveloped. In this work, we provide a practical primer on energy measurements, highlighting key metrics, methodologies, and trade-offs. We introduce a methodology for energy benchmarking, aimed at enabling researchers to incorporate energy considerations into their algorithm evaluations and to design methods explicitly optimized for reduced energy demand. To demonstrate the value of energy-aware benchmarking, we focus on sampling, a fundamental and widely used operation in probabilistic machine learning that often incurs substantial computational cost. Guided by the insights gained from the analysis of algorithmic factors that affect energy efficiency, we introduce a novel sampling method that allows energy savings of up to $17\times$ compared to widely used approaches. Our results show how integrating energy considerations into the design and evaluation of algorithms can enable more sustainable machine learning practices.

1 Introduction

The continued deployment of machine learning (ML) methods in data centers, cloud devices, and user appliances alike is accompanied by increased concerns about the growing energy demand of the field [International Energy Agency, 2025, Gadepally, 2025]. Countermeasures include reducing the carbon-intensity of the electricity supply or shifting training and inference to times or physical locations with a higher share of renewable energy sources [Yang et al., 2023, Wiesner et al., 2023]. However, we argue that reducing the energy demand of the operations themselves is worthwhile.

Current work on efficiency in ML typically prioritizes performance metrics such as execution speed or floating-point operations per second (FLOPs). Yet these metrics are not necessarily correlated with actual energy consumption. Moreover, energy measurements in the field are often superficial, frequently capturing only system-level overhead rather than the true energy cost of algorithmic execution. Although a rich body of literature on energy measurement exists in other domains, the ML community remains comparatively underexposed to rigorous, high-quality energy evaluations [Chung et al., 2025, Różycki et al., 2025]. As a result, algorithms are still predominantly designed with speed and accuracy in mind, but rarely with explicit consideration of energy efficiency.

We argue that energy measurements, when performed correctly, are neither prohibitively difficult nor intrusive. Computing hardware vendors increasingly provision their platforms with power or energy meters. Given minimal methodological understanding, energy measurements can be hence integrated into algorithm evaluations with low overhead. Our goal is therefore to support broader adoption of

*Equal contribution.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: .

energy-aware benchmarking and design in ML by: (i) raising awareness of the issue, (ii) providing a structured overview of relevant energy measurement methodologies, (iii) showcasing an exemplary, exhaustive energy evaluation, and (iv) demonstrating how integrating energy considerations into the design of algorithms can lead to more efficient algorithms. We present a holistic methodology for assessing the energy efficiency of algorithms, aimed at enabling researchers to incorporate energy considerations into their evaluations and to design methods explicitly optimized for reduced energy demand.

As an illustrative application, we focus on the task of *sampling*, a fundamental building block in ML and beyond. Sampling is at the core of numerous widely applied models and algorithms, ranging from Bayesian inference to generative modeling. Despite its conceptual simplicity, the process of generating random variates is often associated with significant computational costs and, consequently, high energy consumption. By introducing a novel sampling algorithm that outperforms state-of-the-art approaches in energy efficiency while demonstrating competitive sampling speed, we showcase how careful analysis of the algorithmic factors that affect the energy efficiency, combined with rigorous energy benchmarking, can lead to the design of novel, more energy-efficient algorithms.

2 Energy Efficiency

Energy efficiency in computing systems is shaped by hardware and algorithmic choices and expressed using various metrics emphasizing different optimization goals. In this work, we begin by defining the key concepts underlying energy-efficient computation (Section 2.1). We then introduce methodologies for measuring energy consumption (Section 2.2 and 2.3), followed by an analysis of common factors that influence the energy efficiency of algorithms (Section 2.4).

2.1 Concepts: What to Measure

Power is defined as the rate at which electricity is consumed, and *energy* is the cumulative amount of electricity required to perform a task. Therefore, energy is power integrated with respect to time $[t_0, t_1]$, as shown in Equation (1). Power ($\mathcal{P}$) is measured in Watts (W) or Joules per second, while energy ($\mathcal{E}$) is measured in Joules (J). In the context of computing, energy is often the more appropriate metric for optimization, as it directly relates to battery lifetime in mobile devices, operating costs in data centers, and environmental sustainability in terms of carbon dioxide emissions.

$$\mathcal{E}[\mathrm{J}] = \int_{t_0}^{t_1} \mathcal{P}[\mathrm{W}](t[\mathrm{s}])\,\mathrm{d}t\,. \tag{1}$$

However, optimizing only for energy may neglect power draw and performance requirements. Peak power draw is important to consider, given the hardware limitations on the power supply or availability in the electricity grid. The power demand is influenced by multiple factors [Kaxiras and Martonosi, 2008]:

$$\mathcal{P} \sim \underbrace{N \cdot C \cdot F \cdot V^2}_{\text{dynamic}} + \underbrace{N \cdot R_{\text{leak}}^{-1} \cdot V^2}_{\text{static}}. \tag{2}$$

Here, N denotes the number of actively switching transistors, which depends on both the circuit design and the program running. C denotes the capacitance, which is given by the circuit design. F denotes the clock frequency, which configurable (impacting performance) but limited by the circuit design. V denotes the supply coltage, which relates to the clock frequency C and configurable as well, but again limited by the circuit design. R_{leak} denotes the leak resistance and is is fixed by the circuit design. Therefore, reducing the power demand can be mainly achieved by adjusting the clock frequency or supply voltage, and through software design, by leveraging computations requiring a lesser number of active transistors.

Optimizing algorithms exclusively for speed can lead to excessive energy consumption, while optimizing only for energy may degrade performance; therefore, the *energy-delay product* (EDP) serves as a balanced optimization metric that accounts for both energy demand and computational speed in algorithm design. *Energy-delay-squared product* (ED^{2}P) prioritizes performance more strongly, as shown in the Equation (3):

$$\mathrm{EDP} = \mathcal{E}[\mathrm{J}] \cdot t[\mathrm{s}], \quad \mathrm{ED}^2\mathrm{P} = \mathcal{E}[\mathrm{J}] \cdot t^2[\mathrm{s}^2]\,. \tag{3}$$

This becomes more pronounced as even within the same system the power demand can change due to effects like dynamic voltage and frequency scaling, for which the ED^2P can be show to be better suited [Brooks et al., 2000].

2.2 Methodologies: How to Measure

Measuring power draw or energy consumption in computing systems can be approached from two complementary perspectives: direct physical measurement and indirect logical modeling.

Physical measurement Physical measurements rely on electrical sensors that directly or indirectly capture voltage and current in circuits. Examples include shunt resistors and Hall effect sensors [Ziegler et al., 2009]. Physical measurement methods provide high accuracy and fine temporal resolution, but require setup alterations in the form of hardware modifications, which may be impractical and sometimes even infeasible.

Logical modeling Alternatively, software power models can estimate power draw and energy consumption by multiplying hardware performance counters (cycles, instruction count, cache accesses etc.) or other software-level statistics with initial physical measurements of single operations [Bellosa, 2000]. For example, instead of measuring the energy consumption of a sampling routine on the circuit level using techniques described above, one counts the operations (lookup, addition, etc.) performed via software and then multiplies by separately taken physical measurements of the individual operations. While easier to integrate as no additional circuits are required, their accuracy depends heavily on the quality of the model (potentially disregarding side-effects such as cooling due to workload), thus making these methods ultimately more error-prone.

2.3 Hierarchy of Measurement Facilities

Another important dimension is granularity. One may analyze energy demand per instruction, per algorithm, or per system. At the instruction level, activity factors such as the number of transistor switches, the clock frequency, and supply voltage determine dynamic power [Kaxiras and Martonosi 2008, see Section 2.1]. At higher levels, interactions with the memory hierarchy, I/O, cooling, and communication can contribute substantially to overall consumption. Analyzing the energy demand at different levels requires different measurement facilities. Extending on existing classification Köhler et al. [2020], the measurement facilities can be categorized on where there are placed in relation to the computing system and how many components they measure at the same time.

External devices The most versatile but also coarse-grained instruments intercept the power supply between the power source and the device under test. Therefore, it yields a comprehensive description of the power draw and energy consumption of execution, but including the energy consumption of other processes going on in the device. Hence, it is advisable to control for confounders like storage activity, network connectivity and display brightness when performing measurements. In this study, we utilize the Microchip MCP39F511N, a dual-channel monitor that can measure the current power draw at up to 240 Hz and a 0.5% value error margin.

On-board monitors Modern mainboards contain power sensors, which are accessible through interfaces like the Intelligent Platform Management Interface (IPMI). Their resolution in time and value is typically not sufficient for machine learning applications. Without a dedicated power meter at the power supply of, e.g., for the GPU, the informative value is not higher than an external measurement device. However, hardware vendors offer development boards with dedicated power rails to report voltage, current, and power draw of different components of the system. For example, boards of the NVIDIA Jetson series integrate measurement facilities for CPU, GPU, the memory subsystem, WiFi, and the entire system. These devices can help with the analysis and optimization of machine learning algorithms, or the formulation of logical energy models. They provide moderate accuracy at little overhead and expose measurements through Linux kernel interfaces.

On-chip counters At the finest granularity, processors may integrate energy monitoring themselves. Intel's Running Average Power Limit [David et al. 2010, RAPL] interface is a well-known example. RAPL registers capture cumulative energy consumption (not power draw) at about 1 ms resolution. RAPL exposes the energy usage per domain (CPU cores, internal GPU, DRAM, full package).

RAPL-compatible interfaces exist in AMD processors since Ryzen Gen 3. Recent Apple Silicon processors (since M1) introduced their own set of hardware counters, including energy counters for the GPU and the Neural Engine. These mechanisms enable detailed energy accounting with negligible overhead, but are restricted to supported platforms.

Likewise, GPU vendors integrate energy or power meters, accessible through their software stack. For example the NVIDIA System Management Interface (`nvidia-smi`) can report the current power draw at a point in time, which can be numerically integrated into the energy demand of an operation.

Best practices Even at fixed clock rates, switching between CPU architectures can significantly alter power demand but not necessarily energy demand. A low-power device (a microcontroller or efficient CPU core) can run for a longer time than a more power-intensive one, resulting in comparable energy integrals—or not, depending on the static power demand and thus energy proportionality (i.e., the degree of proportionality between energy consumption and workload) of the system [Barroso and Hölzle, 2007]. For a fixed problem size, the latter device can switch to idle mode after completion or process more elements for a given unit of energy. Consequently, to obtain more representative measurements, one should fix the CPU frequency and micro-architecture (cores) in experiments.

2.4 Energy-Efficient Design of Algorithms

The power draw and energy consumption of algorithms are shaped by multiple factors. A systematic analysis of these factors allows for a more comprehensive comparison of existing algorithms, provides insight into implementation trade-offs, and ultimately guides the development of novel energy-efficient methods.

Arithmetic operations The energy cost of an arithmetic operation depends on both the type of operation and the precision of the operands. In general, integer arithmetic is cheaper than floating-point arithmetic because floating-point operations require more complex circuitry, including normalization, rounding, and handling of exponents. Therefore, floating point operations require switching of more transistors, thus increasing power draw [Kaxiras and Martonosi 2008, see Equation (2)]. Among arithmetic operations, additions and subtractions are the least expensive, as they only require basic logic for carrying and summing bits. Multiplications are more costly because they involve repeated addition and shifting steps internally, and floating-point multipliers also require exponent and mantissa adjustments. Division and exponentiation are the most expensive operations, often requiring iterative algorithms or table lookups in hardware, which increase both latency and energy consumption [Knuth, 2014]. Reducing precision (e.g., using half-precision or single-precision floats instead of double) can further reduce energy usage while still maintaining acceptable numerical accuracy for many ML tasks [Micikevicius et al., 2018].

Conditional operations Branches dependent on unpredictable outcomes can lead to pipeline stalls and branch mispredictions, which waste energy. Branchless implementations, using arithmetic operations instead of conditionals, may improve both performance and energy efficiency.

Memory access There is a direct connection between the memory access behavior of modern computer systems and their electricity consumption [Horowitz, 2014]. Memory subsystems and CPU caches have long been overlooked in comparison to computational cores but constitute a large portion of active transistors in today's chip designs, leading to higher dynamic power demands. Memory hierarchy interactions are often more energy-intensive than arithmetic operations. This means that, for general-purpose computers, algorithms that trade computation for memory lookups may have slightly worse energy efficiency than plain recomputation. This effect is more pronounced with multiple, nested lookups (also known as *pointer chasing*) because it involves more active transistors, which increases power demand. It also breaks CPU cache locality and access prediction, resulting in prolonged CPU stalls (increased time demand) and thus non-linear increase in energy demand. Optimizing locality is thus crucial.

Overall, designing energy-efficient algorithms requires balancing these factors, often with application-specific trade-offs between quality, reproducibility, and computational constraints. Beyond these general factors, probabilistic ML adds a further source of energy demand: random number generation, which lies at the core of sampling algorithms.

Entropy source Many algorithms require (pseudo-)randomness, making random number generation an additional source of energy consumption. Hardware random number generators can provide high-quality entropy at low latency but often incur notable power costs. In contrast, software-based pseudo-random number generators (PRNGs) are lightweight, reproducible, and often sufficiently accurate for sampling-based algorithms. Nevertheless, the choice of generator matters: certain cryptographically secure or high-quality generators may introduce significant computational overhead. In extreme cases, the entropy source can become a dominant contributor to energy consumption.

3 Sampling

In the following, we first review classical methods for sampling from discrete probability distributions, along with recent state-of-the-art advances. We then introduce a novel sampling method explicitly designed for energy efficiency while maintaining competitive performance. Our approach leverages the factors discussed in Section 2 to systematically reduce energy demand. Finally, in Section 4, we present an experimental evaluation of these methods, illustrating how careful energy-aware analysis can drive algorithmic innovation.

Formally, by sampling from a discrete probability distribution we mean the task of generating random variates from any finite distribution over n outcomes. The distribution is characterized by probabilities $p_1, \ldots, p_n \in [0, 1]$ associated with outcomes $x_1, \ldots, x_n \in \mathcal{X}$, where $\sum_{i=1}^{n} p_i = 1$. We impose no restrictions on the structure of the outcome space $\mathcal{X}$: outcomes may be real numbers, categorical labels, strings, pointers to complex data structures, or any mixture thereof.

3.1 Common Sampling Methods

We chose three discrete sampling methods to benchmark their energy efficiency and compare our own algorithm against: the inversion method, the alias method, and the Fast Loaded Dice Roller (FLDR). The inversion method [Devroye, 2006] is one of the the most widely used sampling method and is integrated into many commonly used libraries. The Alias method [Walker, 1974b] is a more efficient approach to random variate generation, though less commonly employed in standard libraries. FLDR [Saad et al., 2020] is a recently developed method for discrete sampling that is claimed to be faster than the Alias method and other state-of-the-art competitors. Due to space constraints, the details are provided in the Appendix A. Next, we shortly revisit the idea of lookup table-based sampling in order to then introduce our discrete sampling method.

3.2 Lookup Table based Sampling

Naive lookup table sampling constructs a lookup table containing duplicates of each outcome x_i proportional to its probability p_i:

$$\frac{\text{occurrences of } x_i \text{ in table}}{\text{table size}} = p_i. \tag{4}$$

Sampling a random variate then reduces to uniformly selecting a random table index $I \sim \text{Uniform}\{1, \ldots, N\}$ and returning $S = \text{Table}[I]$, where N is the table size, see Figure 1a ("Classic Lookup Table") for an example lookup table.

In practice, memory constraints bound the table size N, limiting representable probabilities to multiples of $1/N$. Approximating probabilities to precision b bits requires quantizing each p_i to $\hat{p}_i = \text{round}(p_i \cdot 2^b)/2^b$, yielding a table of size $N = 2^b$. While approximation error decreases logarithmically with b, memory requirements grow exponentially, making high-precision sampling prohibitive.

3.3 Our Sampling Method

Our approach, which we call *cLUT*, is based on the idea of lookup tables, reusing precomputed results, while conserving memory requirements and memory accesses. Sampling using lookup tables offers multiple benefits (single lookup, no arithmetics, no branching) that enable energy efficiency as detailed above, but suffers from huge memory requirements even for reasonable precision. Sampling from a distribution with 32-bit precision requires storing $2^{32} \approx 4.3$ billion entries (17GB for values

Distribution

x_i	$p(x_i)$
x_1	7/16
x_2	5/16
x_3	3/16
x_4	1/16

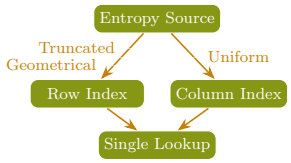
Classic Lookup Table

1/4	x_1	x_1	x_1	x_1
1/4	x_1	x_1	x_1	x_2
1/4	x_2	x_2	x_2	x_2
1/4	x_3	x_3	x_3	x_4
p	1/4	1/4	1/4	1/4

Compressed Table

1/2	x_1	x_2
1/4	x_1	x_3
1/8	x_1	x_2
1/8	x_3	x_4
p	1/2	1/2

(a) Classic and compressed lookup table for example distribution.



(b) Schematic.

Figure 1: (a) The classic and compressed lookup table for an example distribution on given by p (left). The classic lookup table (middle) contains each value according to its probability p at a precision of $b = 4$ bits. The compressed lookup table (right) stores the same distribution. For example, the probability of x_1 is given by the compressed lookup table as $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{1}{2} + \frac{1}{8} \cdot \frac{1}{2} = \frac{7}{16}$. (b) Schematic of our sampling algorithm: A row index is sampled geometrically and a column index uniformly. The sample is generated by a single lookup.

stored in a 32-bit format). The impractical memory requirements (that scale exponentially) motivate our approach to decrease the memory requirements by losslessly compressing the lookup tables strategy, then sampling using the compressed tables with a tailored sampling scheme.

Compression scheme Intuitively, our compressed lookup table can be viewed as a two-dimensional array consisting of $r + 1$ rows and 2^c columns, with $r, c \in \{0, \dots, b\}$ satisfying $2^{r+c} = 2^b = N$. Each row i of the first r rows corresponds to a frequency of 2^{r-i}, where row indices run from 1 to r. The r+1-th row corresponds to the same frequency as the r-th row, namely $2^{r-r} = 1$. For an exemplary compression, see Figure 1a.

This lossless compression scheme preserves the total frequencies while drastically decreasing the size of the lookup table. Compressing a naive lookup table with $N = 2^{r+c}$ entries to a compressed lookup table with $(r + 1) \cdot 2^c$ entries yields a compression ratio of $\rho = 2^r/(r + 1)$.

Sampling step To generate a sample $S \in \mathcal{X}$ using a compressed lookup table, we generate two indices independently: a row index $I \in \{1, \dots, r{+}1\}$ and a column index $J \in \{1, \dots, 2^c\}$. We sample the index I according to a truncated geometric distribution, and the column index J uniformly:

$$\mathbb{P}(I = i) = \max(2^{-i}, 2^{-r}) \quad \text{for } i = 1, \dots, r{+}1, \quad \text{and } \mathbb{P}(J = j) = 2^{-c} \quad \text{for } j = 1, \dots, 2^c.$$

Therefore, we sample a table-index $(I, J) = (i, j)$ with probability $2^{-\min(i,r)-c}$. The column index J can be efficiently sampled using any uniform sampler. The row index I can also be sampled extremely efficiently using, for example, the procedure detailed in Algorithm 1 in lines 2-5. A sample is then generated by returning the value stored in the compressed lookup table at that index:

$$S = \mathcal{T}[I, J].$$

Algorithm 1 Our approach: Sampling using compressed lookup tables

Require: number of samples n, compressed lookup table $\mathcal{T}$ of size $(r + 1) \times 2^c$
Ensure: array of samples $\mathcal{S}$ of size n

1: **for** $k = 1$ **to** n **do**
2: *// Generate a row index geometrically:*
3: $I \leftarrow 1$
4: **while** $I < r + 1$ & `randomBit()` $= 1$ **do**
5: $I \leftarrow I + 1$
6: *// Generate a column index uniformly:*
7: $J \leftarrow \text{Uniform}\{1, \dots, 2^c\}$
8: *// Look up sample at generated index:*
9: $\mathcal{S}[k] \leftarrow \mathcal{T}[I, J]$
10: **return** $\mathcal{S}$

Preprocessing step We construct the compressed lookup table directly from the binary expansion of the probabilities $\hat{p}_i$ rounded to precision of 2^{-b}. A value x_i appears in row j if and only if the

j-th bit of $\hat{p}_i$ is one. The rounded probabilites $\hat{p}_i$ can be adjusted to sum to exactly 1 by using a sum-preserving rounding scheme, making our sampling procedure rejection-free. The number of active bits across the binary representations of the $\hat{p}_i$ may differ, which results in rows of unequal width in the initial construction of the compressed lookup table. To improve the sampling speed, we ensure that all rows have uniform width by moving entries to rows below while doubling them (to ensure correct frequencies).

4 Evaluation

Evaluation setup Our experiments were conducted on a conventional laptop powered by an Intel i7-1255U processor and 16 GiB of DDR4 RAM, running Ubuntu Linux. To ensure accurate measurements, as to prevent inaccuracies from *Dynamic Frequency and Voltage Scaling* [Le Sueur and Heiser, 2010] and thread movement across cores, we fixed the CPU's clock rate at its maximum of 4.7 GHz and bound single-thread workloads to a single core. As our particular Intel *Hybrid* CPU architecture comprises of larger **p**erformance cores and limited **e**fficiency cores, we opted for the P-cores for consistent measurements. We test state-of-the-art approaches and our cLUT method in Python and C. We compare all algorithms on a set of synthetically generated distributions (with the number of unique outcomes n spanning the range $[10^4, 10^7]$), randomly sampled from Dirichlet priors to ensure a wide range of entropy values. We measure energy usage, average power, EDP, and execution time.

Evaluation of Python implementations We benchmark our cLUT algorithm and discrete samplers from popular Python ML libraries, namely, `RandomGenerator.choice()` from NumPy, `multinomial()` from PyTorch, and `random.choice()` from JAX [Harris et al., 2020, Paszke et al., 2019, Bradbury et al., 2018]. The discrete samplers from NumPy, PyTorch and JAX all use (optimized) versions of the inversion method (see Appendix A). To simulate a real-world sampling application, time and energy measurements account for the entire pipeline, including preprocessing, 10^8 sampling iterations, and the execution overhead. To comply with a realistic and user-friendly evaluation scenario, we perform measurements via logical modeling as opposed to direct physical measurements, as described in Section 2.2. For robustness, energy consumption is measured by two sources: RAPL interface as an on-chip counter and Microchip MCP39F511N (MCP) as an external source (see Section 2.3). Average power and EDP values are computed with respect to RAPL data (see Section 2.1). Due to the space restrictions, we provide results only on the package domain of RAPL as the most representative.

Figure 2 shows the mean and standard deviation of energy consumption and execution time for each distribution size. The discrete sampler from NumPy is shown to be the least efficient overall, while JAX and PyTorch exhibit varying relative performance depending on the size of the distribution. The proposed cLUT sampler demonstrates efficiency in all dimensions: the energy demand of cLUT is up to 17$\times$ lower for larger distributions ($n \in (10^6, 10^7]$) and up to 6$\times$ lower for smaller distributions ($n \in [10^4, 10^6]$), as shown in Table 1. One may notice that energy consumption is not always correlated with the execution time. E.g., in the group of larger distributions, JAX exhibits longer runtimes than PyTorch yet consumes less energy. In such scenarios, the Energy-Delay Product (EDP, see Section 2.1) offers a more balanced metric, revealing that PyTorch achieves greater overall efficiency (Table 1).

Evaluation of C implementations As the C implementation allows for low-level energy measurements, we distinguish between the preprocessing and sampling stages and measure them separately. We run 10^3 warm-up sampling iterations before doing actual measurements for 10^7 sampling iterations. As in the previous group of experiments, we measure the package domain from RAPL for energy consumption. MCP measurements are not applicable because the runtime is too short for capturing accurate energy readings. We compare cLUT to C implementations of classic methods and recent state-of-the-art advances for discrete sampling.

Figure 3 illustrates the energy efficiency of the evaluated methods and highlights the systematic advantage of the cLUT sampler in reducing energy consumption. This advantage arises not only from acceleration but also from the elimination of computationally expensive operations; figures are shown in Table 2. The polarity of the power data is explained by the different behavior of the methods on different entropy ranges: distributions with high entropy are more likely to belong to the

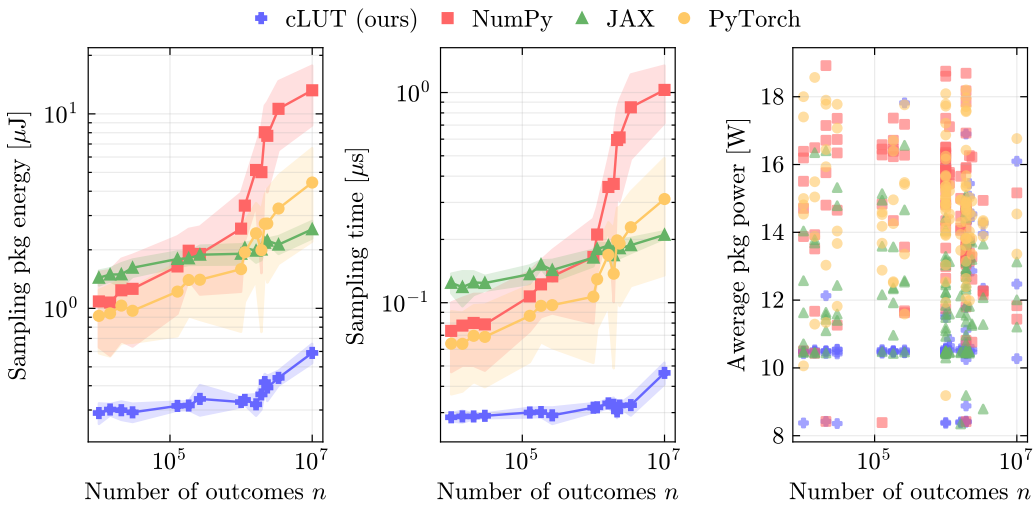


Figure 2: Energy efficiency of built-in discrete samplers from widely used libraries (NumPy, JAX, PyTorch) and our approach (cLUT). Shown are (1) average energy consumption (RAPL:pkg, mean and standard deviation microjoules), (2) the overall execution time (mean and standard deviation in microseconds), and (3) average power draw (each point denotes a result for a distribution, in Watts) per one single sample from distributions of varying sizes n. The plots are shown on a log-log scale.

	Sampler	RAPL:pkg [μJ]	MCP [μJ]	Time [μs]	Average power [W]	EDP [$\mu J \cdot \mu s$]
$n \in [10^4, 10^6]$	JAX	1.76 ± 0.26	3.45 ± 0.47	0.15 ± 0.02	12.17 ± 1.69	0.26 ± 0.07
	NumPy	1.95 ± 1.15	3.34 ± 1.92	0.13 ± 0.07	15.12 ± 2.18	0.33 ± 0.33
	PyTorch	1.33 ± 0.67	2.33 ± 1.17	0.09 ± 0.05	14.86 ± 1.99	0.15 ± 0.14
	cLUT (ours)	$\mathbf{0.32 \pm 0.03}$	$\mathbf{0.62 \pm 0.05}$	$\mathbf{0.03 \pm 0.00}$	$\mathbf{10.50 \pm 1.09}$	$\mathbf{0.01 \pm 0.00}$
$n \in (10^6, 10^7]$	JAX	2.08 ± 0.25	4.09 ± 0.46	0.18 ± 0.02	11.80 ± 1.52	0.37 ± 0.08
	NumPy	6.30 ± 4.00	11.70 ± 7.77	0.47 ± 0.32	14.07 ± 1.93	4.23 ± 4.58
	PyTorch	2.40 ± 1.36	4.29 ± 2.51	0.17 ± 0.10	14.68 ± 1.38	0.53 ± 0.54
	cLUT (ours)	$\mathbf{0.38 \pm 0.08}$	$\mathbf{0.72 \pm 0.13}$	$\mathbf{0.03 \pm 0.00}$	$\mathbf{11.53 \pm 2.20}$	$\mathbf{0.01 \pm 0.00}$

Table 1: Benchmark of discrete samplers from popular Python libraries relevant to machine learning (JAX, NumPy, PyTorch) and our cLUT method. Measured are energy consumption of the sampling procedure (RAPL:pkg, in microjoules), energy consumption of the whole system during execution (MCP, in microjoules), execution time (in microseconds), average power draw (in Watts), and Energy-Delay Product (EDP, see Section 2.1), grouped by the distribution size n (number of unique outcomes). RAPL energy consumption is additionally supplemented by the energy consumption of the overall system measured by MCP. Execution of 10^8 samples was repeated and measured five times, and then averaged to a single iteration.

	Sampler	RAPL:pkg [nJ]	Time [ns]	Average power [W]	EDP [$\mu J \cdot s$]
$n \in [10^4, 10^6]$	Inversion method	728.21 ± 73.56	43.22 ± 2.69	16.99 ± 2.33	31.3 ± 2.16
	Alias method	264.06 ± 118.05	17.39 ± 7.02	15.41 ± 3.66	5.25 ± 4.33
	FLDR	256.22 ± 88.76	16.59 ± 5.0	15.62 ± 2.75	4.61 ± 3.24
	cLUT (ours)	$\mathbf{175.21 \pm 47.32}$	$\mathbf{14.32 \pm 2.48}$	$\mathbf{12.58 \pm 4.05}$	$\mathbf{2.52 \pm 0.92}$
$n \in (10^6, 10^7]$	Inversion method	676.31 ± 110.39	45.07 ± 4.39	15.34 ± 3.52	30.04 ± 3.0
	Alias method	422.41 ± 104.72	34.04 ± 8.12	12.81 ± 3.46	14.75 ± 7.2
	FLDR	278.66 ± 176.24	19.49 ± 10.9	14.37 ± 3.55	7.15 ± 11.7
	cLUT (ours)	$\mathbf{179.41 \pm 64.68}$	$\mathbf{15.53 \pm 4.29}$	$\mathbf{11.7 \pm 3.33}$	$\mathbf{2.97 \pm 2.14}$

Table 2: Benchmark of C implementations of classic methods and recent state-of-the-art advances for discrete sampling, and our proposed cLUT method (see Section 3.3). Measured are average energy consumption (RAPL:pkg, in nanojoules), execution time (in nanoseconds), power draw (in Watts), and Energy-Delay Product (EDP, see Section 2.1) per single iteration of sampling, grouped by the distribution size n (number of unique outcomes). Execution of 10^7 samples was repeated and measured five times, and then averaged to a single iteration.

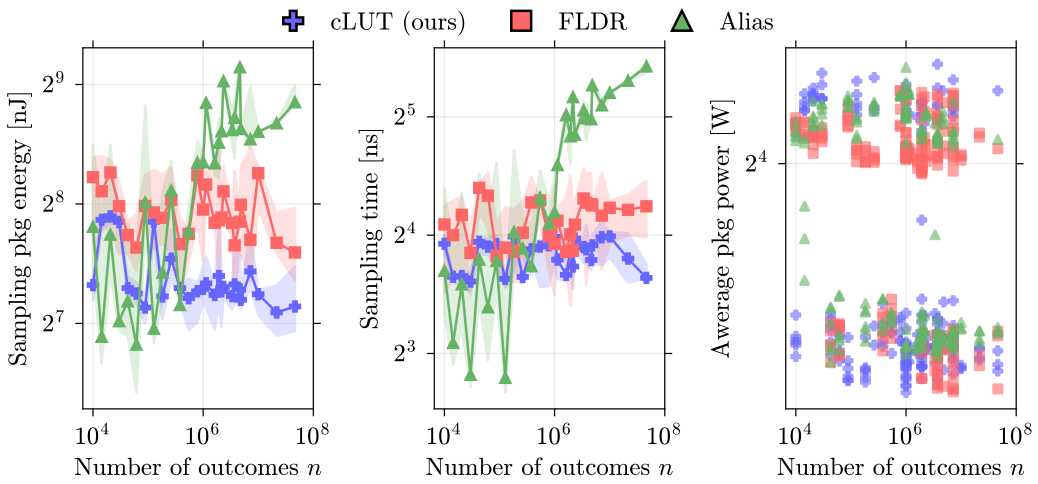


Figure 3: Energy efficiency of C implementation for classic methods and recent state-of-the-art advances for discrete sampling, and our proposed cLUT method. Shown are (1) average energy consumption (RAPL:pkg, mean and the interval between the 1st and 3rd quartiles in nanojoules), (2) execution time (mean and the interval between the 1st and 3rd quartiles in nanoseconds), and (3) average power draw (each point denotes a result for a distribution, in Watts) per one single sample from distributions of varying sizes n. Preprocessing time was measured separately and subtracted to isolate the pure sampling time. The plots are shown on a log-log scale.

upper cluster. The inversion method is shown to be less scalable with respect to the distribution size n, and the inversion method is found to be entirely uncompetitive (Table 2).

5 Conclusion

In this work, we provided a comprehensive primer and framework for incorporating energy awareness into algorithmic research, outlining fundamental concepts, metrics, and methodologies for assessing power draw and energy efficiency. We demonstrated how such analyses can reveal previously hidden trade-offs and guide the design of more sustainable computational methods.

As an illustrative case study, we examined discrete sampling, a fundamental operation in probabilistic machine learning and beyond. Based on rigorous analysis of the energy dynamics of sampling algorithms, we derived our own energy efficient sampling method *cLUT*. Through extensive benchmarking across multiple software frameworks and implementation levels, we showed that cLUT substantially reduces both execution time and energy demand. In Python-level evaluations, cLUT achieved up to a 17$\times$ reduction in energy consumption for large distributions ($n > 10^6$) and up to a 6$\times$ reduction for smaller ones, while maintaining or exceeding the speed of competing implementations in NumPy, PyTorch, and JAX. In low-level C benchmarks, cLUT consistently demonstrated the lowest average power consumption and energy-delay product, confirming that its efficiency stems from the elimination of energy-intensive operations.

These findings emphasize that energy consumption is not necessarily correlated with runtime or throughput alone. Metrics such as the Energy–Delay Product (EDP) provide a more holistic view of efficiency, capturing both computational performance and energy demand. Our experiments highlight that even well-optimized libraries can vary substantially in energy efficiency, underscoring the importance of integrating energy metrics into benchmarking and model evaluation workflows.

Acknowledgments

We are grateful to Toni Mattis (Hasso Plattner Institute) for insightful discussions about energy efficiency that helped to improve this work.

References

Luiz André Barroso and Urs Hölzle. The case for energy-proportional computing. *Computer*, 40(12): 33–37, 2007. doi: 10.1109/MC.2007.443.

Frank Bellosa. The benefits of event: driven energy accounting in power-sensitive systems. In *Proceedings of the 9th workshop on ACM SIGOPS European workshop: beyond the PC: new challenges for the operating system*, pages 37–42, 2000.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

David M Brooks, Pradip Bose, Stanley E Schuster, Hans Jacobson, Prabhakar N Kudva, Alper Buyuktosunoglu, John Wellman, Victor Zyuban, Manish Gupta, and Peter W Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

Jae-Won Chung, Jiachen Liu, Jeff J Ma, Ruofan Wu, Oh Jun Kweon, Yuxuan Xia, Zhiyu Wu, and Mosharaf Chowdhury. The ml. energy benchmark: Toward automated inference energy measurement and optimization. *arXiv preprint arXiv:2505.06371*, 2025.

Howard David, Eugene Gorbatov, Ulf R. Hanebutte, Rahul Khanna, and Christian Le. Rapl: memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED '10, page 189–194, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301466. doi: 10.1145/1840845.1840883.

Luc Devroye. Nonuniform random variate generation. *Handbooks in operations research and management science*, 13:83–121, 2006.

V. Gadepally. Ai has high data center energy costs — but there are solutions, 2025. URL `https://mitsloan.mit.edu/ideas-made-to-matter/ai-has-high-data-center-energy-costs-there-are-solutions`.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020. doi: 10.1038/s41586-020-2649-2. URL `https://doi.org/10.1038/s41586-020-2649-2`.

Mark Horowitz. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE international solid-state circuits conference digest of technical papers (ISSCC)*, pages 10–14. IEEE, 2014.

International Energy Agency. Energy demand from ai, 2025. URL `https://www.iea.org/reports/energy-and-ai/energy-demand-from-ai`.

Stefanos Kaxiras and Margaret Martonosi. *Computer Architecture Techniques for Power-Efficiency*. Springer International Publishing, 2008. ISBN 9783031017216. doi: 10.1007/978-3-031-01721-6. URL `http://dx.doi.org/10.1007/978-3-031-01721-6`.

Donald E Knuth. *The art of computer programming: Seminumerical algorithms, volume 2*. Addison-Wesley Professional, 2014.

Sven Köhler, Benedict Herzog, Timo Hönig, Lukas Wenzel, Max Plauth, Jörg Nolte, Andreas Polze, and Wolfgang Schröder-Preikschat. Pinpoint the joules: Unifying runtime-support for energy measurements on heterogeneous systems. In *2020 IEEE/ACM International Workshop on Runtime and Operating Systems for Supercomputers (ROSS)*, pages 31–40. IEEE, 2020.

Etienne Le Sueur and Gernot Heiser. Dynamic voltage and frequency scaling: the laws of diminishing returns. In *Proceedings of the 2010 International Conference on Power Aware Computing and Systems*, HotPower'10, page 1–8, USA, 2010. USENIX Association.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2018. URL https://arxiv.org/abs/1710.03740.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html.

Rafał Różycki, Dorota Agnieszka Solarska, and Grzegorz Waligóra. Energy-aware machine learning models—a review of recent techniques and perspectives. *Energies*, 18(11):2810, 2025.

Feras Saad, Cameron Freer, Martin Rinard, and Vikash Mansinghka. The Fast Loaded Dice Roller: A Near-Optimal Exact Sampler for Discrete Probability Distributions. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, pages 1036–1046. PMLR, June 2020. URL https://proceedings.mlr.press/v108/saad20a.html. ISSN: 2640-3498.

M.D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering*, 17(9):972–975, September 1991. ISSN 00985589. doi: 10.1109/32.92917. URL http://ieeexplore.ieee.org/document/92917/.

Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974a.

Alastair J Walker. New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10(8):127–128, 1974b.

P. Wiesner, R. Khalili, D. Grinwald, P. Agrawal, L. Thamsen, and O. Kao. Fedzero: Leveraging renewable excess energy in federated learning. *arXiv preprint arXiv:2305.15092*, 2023. URL https://arxiv.org/abs/2305.15092.

Z. Yang, L. Meng, J.-W. Chung, and M. Chowdhury. Chasing low-carbon electricity for practical and sustainable dnn training. *arXiv preprint arXiv:2303.02508*, 2023. URL https://arxiv.org/abs/2303.02508.

Silvio Ziegler, Robert C Woodward, Herbert Ho-Ching Iu, and Lawrence J Borle. Current sensing techniques: A review. *IEEE sensors journal*, 9(4):354–376, 2009.

A Details on sampling algorithms

Alias method The alias method [Walker, 1974a, Vose, 1991] preprocesses a probability distribution over n outcomes into two tables: a probability table and an alias table. Each of the n outcome is represented by an integer between 1 and n (e.g. through the mapping $x_i \mapsto i$) and assigned a "bucket" in with equal weight $1/n$. The probabilities $p_1, \ldots, p_n$ are redistributed so that each bucket contains either a single outcome with probability $p_i \geq 1/n$, or a mixture of two outcomes. Two tables are constructed: the probability table and the alias table. The probability table stores the proportion at which bucket i is "filled" with outcome i. The alias table stores the second outcome, if any, that is distributes to the bucket.

During sampling, one first selects a bucket uniformly at random, and then performs a biased coin flip to decide between the primary and alias outcome stored in that bucket. Sampling occurs therefore in two steps. First, a "bucket" is randomly selected among $\{1, \ldots, n\}$ with equal probability $(1/n)$. Second, the value stored in the probability table in that bucket is read out and a biased coin is flipped according the read probability, e.g., by generating a uniform random variate on the unit interval $[0, 1]$ and checking wether it is bigger than the specified probability. If heads, the bucket index is returned; if tails, the value stored at the Alias table is returned. This has the time complexity of generating a single sample at $O(1)$ after $O(n)$ preprocessing, at $O(n)$ memory. See Figure 4 for an illustration. Figure 4 contains one possible implementation of the alias method for a distribution with $n = 5$ outcomes given by the probabilities $(7/25, 6/25, 1/25, 6/25, 1/5)$. Generation of a single sample could look like this: (i) a random bucket index is drawn equiprobable and turns out to be 4, (ii) a biased coin is flipped with $4/5$ probability to land heads. It lands heads and 4 is returned. If it would have landed tails, 2 would have been returned, since 2 is stored in the alias table at bucket 4.

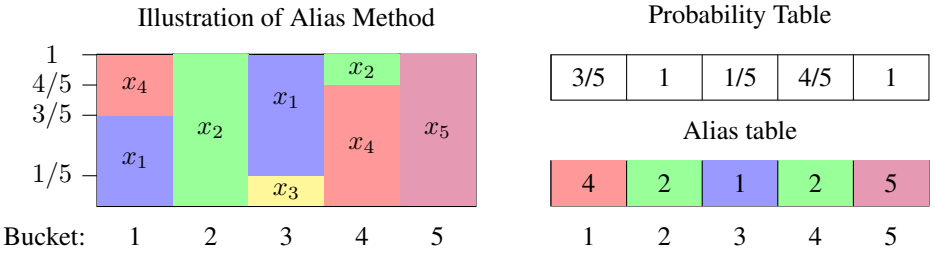


Figure 4: The Alias method for sampling from a discrete distribution. Each bucket has equal probability $1/n$. A coin flip decides between the primary and alias outcome. Example of a distribution with $n = 5$ outcomes given by the probabilities $(7/25, 6/25, 1/25, 6/25, 1/5)$.

Inversion method The inversion method relies on the cumulative distribution function (CDF) $F(x)$ associated with the distribution. Given a uniform random variate $U \sim \mathrm{Unif}(0, 1)$, one obtains a sample from the distribution by computing (see Figure 5 for an illustration):

$$S = F^{-1}(U)\,.$$

This construction works because the CDF transforms the probability space into the unit interval, and applying the inverse recovers the correct distribution of outcomes. The method is conceptually simple, but requires efficient computation of the inverse CDF, which is only analytically available for certain distributions (e.g., exponential, geometric). For more complex distributions, numerical approximation or interpolation of the inverse CDF may be used, at the expense of computational and energy cost.

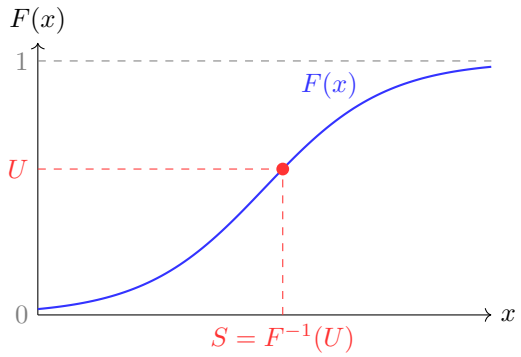


Figure 5: Illustration of the inversion method. A uniform variate $U \sim \mathrm{Unif}[0, 1]$ is drawn, then mapped through the inverse CDF F^{-1} to obtain a sample S that is distributed according to F.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The claims made match the results of the evaluation section.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, area of application of different metrics and devices is discussed.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: The paper is mostly focused on empirical measurements.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Both the algorithm as well as the evaluation setup are detailed extensively, and the code is provided.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Code and data are provided.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: In the evaluation setup section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Standard deviations are provided and mentioned in the tables.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Detailed in the evaluation setup section.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: To the best of our knowledge, there are no deviations from the Code of Ethics, and anonymity is preserved.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The environmental effect of inefficient sampling algorithms is discussed in the introduction, and the potential for reduction in energy consumption is clearly stated.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: There is no risk, as the sampling yields the same results as previous exact samplers, just more efficient.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Authors are credited in code and manuscript.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Details are communicated in the paper.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Not applicable.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: LLMs not involved.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.