Statistical Robustness of Markov Chain Monte Carlo Accelerators

Xiangyu Zhang
Duke University
Durham, North Carolina, USA
xiangyu.zhang@duke.edu

Ramin Bashizade
Duke University
Durham, North Carolina, USA
ramin@cs.duke.edu

Yicheng Wang
Duke University
Durham, North Carolina, USA

Sayan Mukherjee
Duke University
Durham, North Carolina, USA
sayan@stat.duke.edu

Alvin R. Lebeck
Duke University
Durham, North Carolina, USA
alvy@cs.duke.edu

ABSTRACT

Statistical machine learning often uses probabilistic models and algorithms, such as Markov Chain Monte Carlo (MCMC), to solve a wide range of problems. Probabilistic computations, often considered too slow on conventional processors, can be accelerated with specialized hardware by exploiting parallelism and optimizing the design using various approximation techniques. Current methodologies for evaluating correctness of probabilistic accelerators are often incomplete, mostly focusing only on end-point result quality ("accuracy"). It is important for hardware designers and domain experts to look beyond end-point "accuracy" and be aware of how hardware optimizations impact statistical properties.

This work takes a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators. We propose three pillars of statistical robustness: 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit. We apply our framework to a representative MCMC accelerator and surface design issues that cannot be exposed using only application end-point result quality. We demonstrate the benefits of this framework to guide design space exploration in a case study showing that statistical robustness comparable to floating-point software can be achieved with limited precision, avoiding floating-point hardware overheads.

CCS CONCEPTS

• **Computer systems organization** → **Special purpose systems**; • **Hardware** → **Robustness**; • **General and reference** → **Evaluation**.

KEYWORDS

accelerator, statistical machine learning, probabilistic computing , statistical robustness, markov chain monte carlo

ASPLOS '21, April 19–23, 2021, Virtual, USA

ACM ISBN 978-1-4503-8317-2/21/04…$15.00
https://doi.org/10.1145/3445814.3446697

ACM Reference Format:
Xiangyu Zhang, Ramin Bashizade, Yicheng Wang, Sayan Mukherjee, and Alvin R. Lebeck. 2021. Statistical Robustness of Markov Chain Monte Carlo Accelerators. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '21), April 19–23, 2021, Virtual, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3445814.3446697

1 INTRODUCTION

Statistical machine learning, like other methods in artificial intelligence, has become an important workload for computing systems. Such workloads often utilize probabilistic computing, including probabilistic models and probabilistic algorithms, which enable the potential to provide generalized frameworks to solve a wide range of problems. As alternatives to Deep Neural Networks, probabilistic computing provides easier access to interpreting why a given result is obtained due to model transparency and measurable statistical properties. Many specialized accelerators propose to address the sampling inefficiency of probabilistic algorithms (e.g., Markov Chain Monte Carlo Bayesian Inference), by utilizing approximation techniques to improve the hardware efficiency, such as reducing bit representation, truncating small values to zero, or simplifying the random number generator.

Understanding the influence of these approximations on application results is crucial to meet the quality requirement. A hardware accelerator should provide correct execution of target algorithms. A common approach to evaluating correctness is to compare the end-point result quality, such as prediction accuracy, against accurately-measured or hand-labeled ground-truth data using community-standard benchmarks and metrics: the hardware execution is considered to be correct if it provides comparable prediction accuracy to the baseline implementations that do not have the specific approximations.

Although statistical guarantees can be made regarding end-point results [55, 63], domain experts in statistics are interested in the full distribution of possible results rather than a single-point estimate. For example, the outcome of Bayesian Inference is a posterior distribution that includes both the end-point result and uncertainty. The end-point result can be obtained by finding the mode or mean of the distribution. Instead of only knowing the class of an object in an image, statisticians also want to know the probabilities of the object belonging to other classes (*aleatory uncertainty*) and the

overall confidence on the result distribution (*epistemic uncertainty*, or "*uncertainty on uncertainty*" [78]).

Quantifying uncertainty plays a critical role in probabilistic machine learning and failure to adequately or appropriately consider uncertainty information can lead to adverse or catastrophic outcomes, such as a surgeon failing to completely remove a tumor due to incorrect uncertainty in a segmented image [12, 60], or a self-driving car leading to the death of a driver due to over-confidently recognizing a truck as the sky [78]. Furthermore, measuring end-point results may not always be possible since ground-truth data is not always accessible. Therefore, statements and guarantees on the application end-point results are necessary but not sufficient to claim correctness. Probabilistic accelerators should produce correct uncertainty along with correct end-point results.

Unlike end-point results, uncertainty is usually difficult to directly present and can be intractable to compute for high dimensional problems. Instead, domain experts use *statistical robustness* to indicate how faithfully software/hardware produces result distributions, which can be obtained using a collection of statistical metrics. Good statistical robustness often indicates high confidence regarding the produced distribution and the learning/inference method is more robust to perturbed inputs. In the domain of probabilistic computing, correctness is defined by more than the end-point result of executing the algorithm, and includes statistical robustness.

Current methodologies for evaluating probabilistic accelerators are often incomplete or ad hoc in evaluating correctness, focusing only on end-point results or limited statistical properties. Therefore, *a probabilistic architecture should provide some measure (or guarantee) of statistical robustness.* This work takes a first step toward defining metrics and a methodology for quantitatively evaluating correctness of probabilistic accelerators beyond just end-point result quality.

We use Markov Chain Monte Marlo (MCMC) accelerators as a case study, and propose three pillars of statistical robustness: 1) *sampling quality*, 2) *convergence diagnostic*, and 3) *goodness of fit* **(Contribution 1)**. Each pillar has at least one quantitative empirical metric: Effective Sample Size (ESS) for sampling quality; Gelman-Rubin's $\hat{R}$ and convergence percentage for convergence diagnostic; and Root Mean Squared Error (RMSE) and Jensen-Shannon Divergence (JSD) for goodness of fit. These pillars do not require ground-truth data and collectively enable comparison of specialized hardware to a target precision, such as a 64-bit floating-point (FP64) software implementation. We expose several challenges with naively applying existing popular metrics for our purposes, including high dimensionality of the target applications and random variables with zero variance. Therefore, we modify the existing methodologies for sampling quality and convergence diagnostic, and propose a new metric (convergence percentage) for convergence diagnostic **(Contribution 2)**.

The three pillars of statistical robustness can inform end-users by characterizing existing hardware and inform hardware designers by using the pillars in design space exploration. As a case study, we demonstrate the framework using a representative MCMC accelerator—Stochastic Processing Unit (SPU) [87]—and show that end-point result quality alone is insufficient to claim correctness: the accelerator with limited precision and other aggressive approximation techniques achieves the same application end-point result quality as FP64-software, confirming the previous work, but has compromised ESS and convergence percentage results **(Contribution 3)**. The analysis reveals that applications need to run 2× more iterations on the accelerator to achieve the same statistical robustness as FP64-software, reducing the accelerator's effective speedup. We also demonstrate the benefits of using this framework on design space exploration to overcome the limitation of the above accelerator **(Contribution 4)**. We find that considerable improvement in statistical robustness, comparable to FP64-software, can be achieved by slightly increasing the bit precision from 4 to 6 and removing an approximation technique, with only 1.20× area and 1.10× power.

We believe our work provides a template for analyzing statistical robustness on other accelerators. The remainder of this paper is organized as follows. Section 2 provides the necessary background and motivation for this work. The detailed description of the three pillars is in Section 3. Section 4 describes the analysis of statistical robustness on a representative accelerator and we perform design space exploration using the three pillars in Section 5. Section 6 discusses generality of the proposed methodology, limitations, and future work. Related work is presented in Section 7. Section 8 concludes the paper.

2 BACKGROUND AND MOTIVATION

2.1 Probabilistic Computing and Uncertainties

Probabilistic (stochastic) computing is a powerful approach used in many applications (e.g., image analysis [25], robotics [33], natural language processing [19], global health [32], and wireless communications [34]), which includes probabilistic models and probabilistic algorithms. Probabilistic models, such as Markov Random Fields and Bayesian Networks, represent problems as parameterized probability distributions. Probabilistic algorithms, such as MCMC and Hamiltonian Monte Carlo [62], solve problems by randomly inferring the parameters in the probabilistic models to explain observed data. Probabilistic algorithms create opportunities to provide generalized frameworks and are the only practical approach to solve certain problems, such as high-dimensional inference. Compared with Deep Neural Networks, probabilistic models are conceptually simple, compositional, and interpretable [27]. As a key feature, probabilistic computing not only produces a single-point estimate (a.k.a., the end-point result) but also a full distribution of possible outcomes: the approach not only predicts the outcome of a coin toss but also provides the probability of getting a "head".

Bayesian Inference is an important framework in probabilistic computing, which updates the probability estimate (a.k.a., posterior distribution) for a hypothesis by combining information from prior beliefs and observed data. Suppose X is the latent random variable of interest. The goal is to retrieve the posterior distribution $p(X|data)$ given the prior beliefs of X and the observed $data$ using Bayes' theorem: $p(X|data) \propto p(data|X)p(X)$, where $p(X)$ is the prior distribution of X, and $p(data|X)$ is the likelihood of observing $data$ given a certain value of X. Both X and $data$ can be scalars or multidimensional vectors.

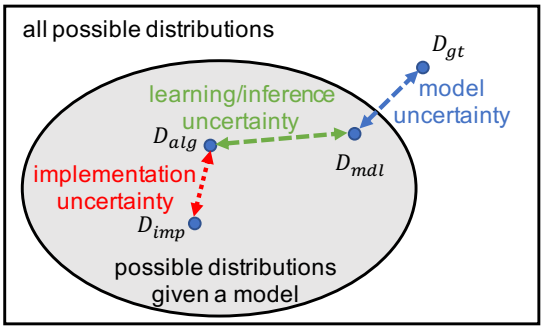


Figure 1: Different types of uncertainties adapted from Hüllermeier and Waegeman [37]. We introduce implementation uncertainty apart from prior work.

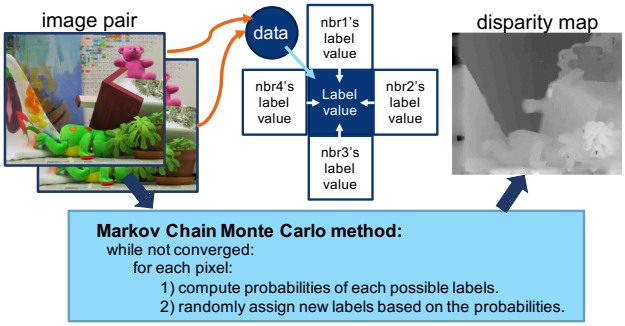


Figure 2: Stereo vision using MCMC (MRF Gibbs Sampling).

As stated by Ghahramani [27], "uncertainty plays a fundamental part" in probabilistic machine learning. Domain experts in statistics, especially in Bayesian Inference, look beyond the end-point result and are also interested in the uncertainty of the produced result, such as the probabilities of other possible outcomes and the overall confidence in the produced result. Hüllermeier and Waegeman wrote a comprehensive introduction to uncertainty in machine learning [37]. One way to categorize uncertainty is by its reducibility. Aleatory uncertainty is due to the inherent randomness of a problem and cannot be reduced. For example, the uncertainty of a coin toss outcome always exists no matter how many data points or how powerful of a predictor we have. Epistemic uncertainty, as Varshney and Alemzadeh also call it "uncertainty on uncertainty" [78], is due to the lack of knowledge including limited model capability, sub-optimal learning/inference algorithms, an inadequate amount of data, insufficient precision in an implementation, etc. Epistemic uncertainty can be reduced by improving the models, algorithms, implementations, and/or collecting more data: we can obtain a better idea of the ratio of "head" vs. "tail" with more coin tosses. Statistical robustness (defined in Section 1) evaluates epistemic uncertainty, and is the focus of this paper. Aleatory and epistemic uncertainties are context-dependent and not absolute notions, as noted by Hüllermeier and Waegeman.

Uncertainty can be further categorized by its source. Figure 1 illustrates the types of uncertainty from different sources. Suppose D_{gt} is the ground-truth distribution—the ground-truth solution of a problem. D_{mdl} is the best possible solution that can be learned/inferred given a probabilistic model (a.k.a., hypothesis space). The discrepancy depicts *model uncertainty* since the model may not be capable of representing the ground-truth solution. D_{alg} is the learned/inferred solution from a probabilistic algorithm given observed data. The discrepancy between D_{mdl} and D_{alg} demonstrates *learning/inference uncertainty*[1] caused by algorithms not finding the optimal solution due to either the algorithm per se or not enough observed data.

The above uncertainties are categorized by prior work [37] related to the selection of models/algorithms and the collection of data. These uncertainties are independent of the software/hardware implementations: the uncertainties exist even with FP64 precision and perfect random number generation. In practice, software or hardware implementations often utilize approximation techniques to improve efficiency. Software implementations can use reduced floating-point representations such as TensorFloat-32 [45] or BFloat16 [82]. Hardware accelerators (see Section 2.3) often utilize limited precision, simplified pseudo-random number generators, or other custom techniques. Intuitively, these approximations potentially introduce epistemic uncertainty: for example, insufficient precision can reduce the knowledge obtainable from the numerical value of input data or intermediate results. Therefore, we introduce *implementation uncertainty*, in addition to model and learning/inference uncertainty, to represent the uncertainty introduced by software/hardware implementations. A good software/hardware implementation should not impose significant uncertainty in the results. The goal of this paper is to quantify implementation uncertainty using statistical robustness metrics.

The rest of this section introduces the performance bottleneck of probabilistic computing using MCMC as an example, the existing probabilistic accelerators that address the bottleneck, and a representative MCMC accelerator that implements various approximation techniques.

2.2 MCMC and Sampling Overhead

In Bayesian Inference, as the dimension of data and probability distribution increase, analytically or numerically deriving the exact result of a posterior distribution becomes complicated and intractable. One approach to breaking "the curse of dimensionality" is Markov Chain Monte Carlo (MCMC), a collection of methods that solve the problems by iteratively generating samples on random variables and eventually converge to a result regardless of the initial stage.

Figure 2 shows an example using an MCMC method, Markov Random Field (MRF) Gibbs Sampling for stereo vision, proposed by previous work [6]. Stereo vision reconstructs the depth information of objects in an image pair by matching the corresponding pixels that represent the same objects. The results are presented in a disparity map, indicating the depth of the corresponding objects in the image (lighter is closer). As shown in Figure 2, the MCMC method iterates the image pixels by considering the disparity of each pixel as a random variable. For each pixel, it evaluates probabilities of each possible label (disparity outcome) and draws a sample as the

[1]The original literature [37] refers to "learning/inference uncertainty" as "approximation uncertainty". We rename it to avoid the confusion with "hardware approximation" which may cause implementation uncertainty in our context.

output label. Each probability is determined by the label values of neighboring pixels and the initial pixel data values of the image pair, defined by First-Order MRF probabilistic graphical model [25]. The outer loop (a.k.a., iteration) iterates on the whole image until convergence.

MCMC methods rely on efficiently performing sampling in the inner loop, which involves step 1) efficiently computing the parameters of the distribution to sample from based on the observed data, and step 2) efficiently generating samples from the parameterized distribution. Unfortunately, as described in previous work [83], sampling overhead can be notably high: step 2) alone takes hundreds of CPU cycles for a simple distribution. One approach to reducing the sampling overhead is to use approximation techniques in algorithms [58, 62, 85]. Deterministic methods, such as Expectation Propagation and Variational Bayesian, are alternatives to MCMC. Although these methods are often more efficient in the applied cases, domain experts use the original MCMC as a conceptually straightforward, mathematically simple, yet accurate framework. As a result, even with sampling overhead, MCMC is one of the most popular algorithms in probabilistic computing. Other examples of MCMC applications include: Herschlag et al. quantify gerrymandering in North Carolina using MCMC sampling [35]; various researchers propose using MCMC as the inner loop for learning different types of Markov Networks [26, 76], which can be used for electronic health records analysis [5].

2.3 Specialized Probabilistic Accelerators

Meeting the needs of domain experts to overcome sampling overhead may be achieved by accelerating sampling through hardware specialization. Previous work seeks to efficiently generate specific types of distributions using FPGAs [1, 74] and specialized circuits [10]. Specialized architectures are proposed to accelerate specific algorithms and models, such as dedicated MCMC accelerators [17, 47, 53, 57, 61, 73, 88], a Bayesian Neural Networks accelerator [9], an accelerator for Stochastic Gradient Descent [54], an ASIC accelerator for Bayesian Networks [44], CMOS-hybrid MRF Gibbs Sampling Unit [83, 87], and compiler workflows to build specialized accelerators [4]. Many of these accelerators use various forms of optimizations (e.g., limited bit precision, pseudo-random number generators, etc.) to reduce area/power, allowing more individual units on a single chip and thus improving overall performance. As described above, *it is important to measure (or guarantee) the correctness of these accelerators, including both end-point result quality and statistical robustness.*

This paper represents a first step toward articulating a set of metrics and methodology for quantifying the statistical robustness of probabilistic accelerators. Section 3 presents our proposed methodology and we use an accelerator [87] (described below) as a case study to demonstrate how to analyze an existing accelerator and to perform design space exploration.

2.4 A Representative Probabilistic Accelerator

As a case study, we consider the Gibbs Sampling accelerator design described by Zhang, et al. [87] implemented entirely in CMOS using pseudo-random number generation instead of molecular optical devices (cf. [87] Section IV). Figure 3 shows the baseline pipeline

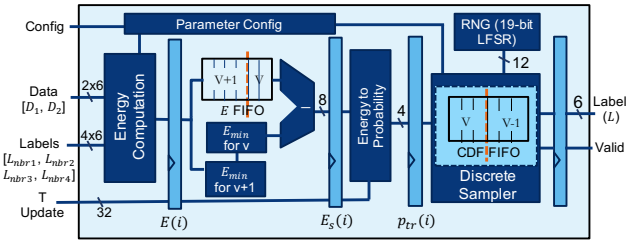


Figure 3: The SPU pipeline derived from RSU-G [87].

design, which we call a Stochastic Processing Unit (SPU). It is divided into four main stages with two internal decoupling FIFOs and an inverse transform method is used for the discrete sampler.

Given the data and neighbor labels, the first stage computes the total energy of a possible label $E(i)$ (Equation 1) each cycle, where α and β are application parameters. The energy $E(i)$ is then dynamically scaled using subtraction to maximize the dynamic range (Equation 2). Both $E(i)$ and $E_s(i)$ are 8-bit unsigned integers. In the third stage, the scaled energy $E_s(i)$ is converted to a scaled probability represented in a 4-bit unsigned integer. The original probability is computed by $exp(-E_s(i)/T)$ which is represented as a real number between (0,1] using floating-point in software, where T is a fixed parameter per outer iteration. However, the probability is scaled using Equation 3 and truncated using Equation 4 to match the unsigned integer representation, where $P_{bits} = 4$ is the number of bits in the scaled probability output $p_{tr}(i)$. Additionally, probability truncation drives all scaled probabilities that are less than one to zero and 2^n approximation rounds all scaled probabilities down to the nearest 2^n integer value (Equation 4). The value of $p_{tr}(i)$ can be pre-computed and stored in a look-up table (LUT). The values in the LUT need updates if T changes between iterations. The final stage of SPU generates a discrete sample per variable based on the probabilities of all possible label values $\{p_{tr}(0), p_{tr}(1), ...\}$ using the least 12-bits of a 19-bit LFSR to implement the inverse transform sampling.

$$E(i) = \alpha E_{singleton}(i) + \beta \sum E_{neighborhood} \quad (1)$$

$$E_s(i) = E(i) - E_{min} \quad (2)$$

$$p_s(i) = (2^{P_{bits}} - 1) \times exp(-E_s(i)/T) \quad (3)$$

$$p_{tr}(i) = \lfloor 2^{\lfloor \log_2 p_s(i) \rfloor} \rfloor \quad (4)$$

The SPU supports two operating modes: 1) pure sampling and 2) optimization (simulated annealing). Pure sampling iteratively generates Gibbs samples using constant temperature T, where T is considered a parameter obtained during model training. When converged, the estimated distribution of a random variable (e.g., distribution of possible disparities in a pixel) can be obtained by collecting the latest N samples. An exact result can be obtained from the mode of the estimated distribution, the most frequent label. The optimization mode uses simulated annealing to converge to an exact result faster by strategically decreasing the temperature T [25]. T is initially high so that each label has a similar probability of being selected. As T decreases, labels with the lowest energy are more likely to be selected, eventually leading to convergence.

Table 1: SPU result quality from one run per dataset.

Motion estimation[1]			Stereo vision[2]		
Dataset	Software	SPU	Dataset	Software	SPU
dimetrodon	0.600	0.611	*art*	26.8%	27.7%
rubberwhale	0.371	0.376	*poster*	12.3%	11.0%
venus	0.460	0.449	*teddy*	26.9%	27.8%

[1] Metric is end-point error [2]. Lower is better.
[2] Metric is bad-pixel percentage [70]. Lower is better.

The optimization mode requires fewer iterations than pure sampling mode, but cannot provide an estimated distribution which is required for uncertainty. Previous work [87] evaluates only the optimization mode.

We implement the SPU in Verilog and use QuestaSim simulation to evaluate the end-point result quality of the same three applications assessed in previous work [87]: image segmentation, motion estimation, and stereo vision. Table 1 shows the result quality comparison between FP64-software and the SPU. Each result is collected by a single run per dataset in optimization mode. We validate that the SPU with a simple 19-bit LFSR as its random number generator (RNG) achieves the same result quality as the software. Image segmentation results indicate the same conclusion and are omitted for brevity. We also obtain similar high-quality application results on an Intel Arria 10 FPGA prototype. Despite these results, we are left with the question: *What do the results in Table 1 indicate about the accelerator's statistical robustness*? The answer is *nothing*. The following sections present our initial efforts toward providing a better answer.

3 THREE PILLARS OF STATISTICAL ROBUSTNESS

To identify appropriate measures of hardware statistical robustness, we draw on known techniques utilized by domain experts to evaluate their models and algorithms. In this work, we focus on MCMC where we view it as a good starting point since domain experts have wide agreement on its statistical robustness metrics. Ideally, we could formally prove bounds on relevant metrics [23, 41]. Unfortunately, some hardware optimizations (e.g., truncation to zero) make formal proofs extremely difficult or impossible. A provable architecture introduces more complicated hardware.

Therefore, we rely on existing empirical diagnostic tests for MCMC techniques, based on foundations in statistics, to establish three pillars for assessing a probabilistic accelerator's statistical robustness: 1) sampling quality [75], 2) convergence diagnostic [16], and 3) goodness of fit. Each pillar has at least one quantitative measure, and provides insight to application users and hardware designers. Compared with end-point result quality, which only measures the difference between a produced distribution mode and some measure of application ground-truth, the proposed pillars collectively help in understanding/explaining end-point results by providing confidence and uncertainty information, and can indicate the performance of the MCMC execution from different aspects: 1) sampling quality measures the rate of generating independent (non-correlated) samples from the target distribution in terms of iterations; 2) convergence diagnostic measures the confidence that the execution has arrived at a stable distribution as the solution; 3) goodness-of-fit measures the difference between a target distribution and the distribution obtained as a solution. The initial metrics we select for each pillar are well-accepted by domain experts. An extensive introduction to MCMC diagnostic metrics is provided by Robert and Casella [66] (Chapter 12). Note that we do not claim completeness, but we believe the pillars are necessary and comprehensively cover key aspects of probabilistic algorithms—a crucial first step.

Recall we are interested in implementation uncertainty introduced by hardware accelerators. Statistical robustness is affected by all uncertainties (i.e., model, learning/inference, and implementation). Therefore, we compare hardware accelerator results with FP64-software as the baseline to isolate the impact of hardware optimizations. The remainder of this section presents our proposed three pillars of statistical robustness.

3.1 Pillar 1: Sampling Quality

A sampling algorithm with perfect sampling quality generates independent samples. However, an MCMC sample is drawn based on the current values of random variables—the outcome of samples in the previous iteration. This dependency creates correlations between samples which is non-trivial until several subsequent samples are drawn, which can be represented as an autocorrelation time τ. This implies that by generating n samples from MCMC, only n/τ samples can be considered independent. A sufficient number of independent samples are required to derive meaningful statistical measures (e.g., mean and variance). Note that the sample dependency is an intrinsic property of MCMC algorithms and exists even with a perfect random number generator and FP64 precision.

Effective Sample Size (ESS) is commonly used as a sampling quality metric that represents how many independent samples are drawn among all the dependent samples. In general, higher ESS indicates the MCMC sampler is more efficient at generating independent samples. Unfortunately, there is no consensus on a single ESS definition [28]. We use the definition discussed by Kass et al. [42] based on autocorrelation time. An alternative ESS definition is "in the custom of survey sampling" [28, 46]. Since closed form expressions for ESS are difficult, we estimate ESS using the known initial positive sequence (IPS) methods [53, 75].

$$ESS = n/(1 + 2\sum_{k=1}^{K} \rho(k)) \tag{5}$$

We estimate ESS on a univariate random variable using Equation 5, where n is the number of dependent MCMC samples (iterations) and $\rho(k)$ is the autocorrelation function of the sample sequence. We sum up the first K contiguous lags where $\rho(k) + \rho(k+1) \geq 0$. Theoretically, $ESS = n$ provides the best sampling quality where all samples are independent; however, Equation 5 is an estimate of ESS, and thus it is numerically possible that $ESS > n$.

The above ESS method cannot be directly applied to our evaluation for two reasons. First, many MCMC problems are high-dimensional (many random variables). For example, in stereo vision a 320×320 input image has 102,400 dimensions. The above ESS does not account for multidimensional problems. Furthermore, the above

ESS has no definition when all collected samples have the same value (zero empirical variance), which is possible in practice as shown in Section 4. An ideal metric can report a scalar ESS value to account for both issues. While methods exist to report multivariate ESS [79], to our knowledge they are not practical in our case and they do not allow zero variance for any variable.

To address multi-dimensionality, we consider each dimension (each pixel in our applications) as a separate random variable (RV) to compute ESS per dimension separately and report a scalar value: the mean ESS of all dimensions. To further address zero variance, we propose two metrics: 1) mean *"overall" ESS* that omits the random variables with zero variance in software and hardware implementations, respectively; and 2) mean *"active" ESS*, a paired metric that only includes the random variables with non-zero variance in both software and hardware. Section 4 suggests overall ESS is biased toward software due to small but non-zero variance. Active ESS omits small variance in software, which can potentially benefit hardware implementations. Algorithms 1 and 2 outline the procedure of computing the two metrics.

Pillar Insight. If ESS is low it may take more MCMC iterations to achieve an acceptable ESS. If a hardware accelerator produces substantially lower ESS than software, the additional iterations may reduce its effective speedup.

Algorithm 1: Overall ESS

Input: trace of multidimensional samples $\boldsymbol{X}$ from a MCMC run, from software or hardware implementations.
sum_ESS = 0, num_valid_rvs = 0
for *x (trace of each univariable RV) in* $\boldsymbol{X}$ **do**
  if *variance(x)* $\neq$ *0* **then**
    sum_ESS += ESS(x)
    num_valid_rvs++
  end
end
Output: overall_ESS = sum_ESS/num_valid_rvs

Algorithm 2: Active ESS

Input: trace of multidimensional samples $\boldsymbol{X_{sw}}$ from a MCMC run in software, and $\boldsymbol{X_{hw}}$ in hardware implementations.
sum_ESS_sw = 0, sum_ESS_hw = 0
num_valid_rvs = 0
for x_{sw} *(trace of each univariable RV in software) in* $\boldsymbol{X_{sw}}$ *and* x_{hw} *(trace of corresponding RV in hardware) in* $\boldsymbol{X_{hw}}$ **do**
  if *variance(*x_{sw}*)* $\neq$ *0 and variance(*x_{hw}*)* $\neq$ *0* **then**
    sum_ESS_sw += ESS(x_{sw})
    sum_ESS_hw += ESS(x_{hw})
    num_valid_rvs++
  end
end
active_ESS_sw = sum_ESS_sw/num_valid_rvs
active_ESS_hw = sum_ESS_hw/num_valid_rvs
Output: active_ESS_sw, active_ESS_hw

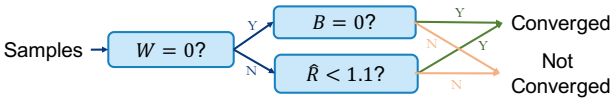


Figure 4: Determine convergence of a random variable.

3.2 Pillar 2: Convergence Diagnostic

An important question for an MCMC method is when to stop iterating, determined by when the MCMC is converged. Similar to ESS, the time to convergence is used to analyze algorithms and input data when using software even with FP64 and sophisticated random number generators. Multiple methods exist to measure the convergence. A comprehensive review is provided by Cowles and Carlin [16]. We use Gelman-Rubin's $\hat{R}$ [8, 24], a popular quantitative method provided by many statistical packages, to measure whether a univariate random variable (e.g., a pixel in stereo vision) is converged at a certain iteration.

Gelman-Rubin's $\hat{R}$ (potential scale reduction factor) estimates the convergence by comparing the between-chain variance (B) and within-chain variance (W) across multiple independent runs on the same MCMC instance[2]. Equations 6 to 9 show the computation to obtain an $\hat{R}$ given the sample trace x from m independent MCMC runs, each with n samples. σ_+^2 is an overestimate on the variance of a random variable. As a rule of thumb [8, 24], a univariate random variable is considered converged when $\hat{R} < 1.1$. Typically larger $\hat{R}$ indicates that more iterations are needed to converge. Note that the $\hat{R}$ method requires a random value initialized from an overdispersed distribution. We meet this requirement by initializing random variables (i.e., initial labels of pixels) uniform-randomly.

$$B/n = \frac{1}{m-1}\sum_{j=1}^{m}(\bar{x}_{j\cdot} - \bar{x}_{\cdot\cdot})^2 \tag{6}$$

$$W = \frac{1}{m(n-1)}\sum_{j=1}^{m}\sum_{t=1}^{n}(x_{jt} - \bar{x}_{j\cdot})^2 \tag{7}$$

$$\hat{\sigma}_+^2 = (n-1)/n \times W + B/n \tag{8}$$

$$\hat{R}^2 = \frac{m+1}{m}\frac{\hat{\sigma}_+^2}{W} - \frac{n-1}{mn} \tag{9}$$

A scalar convergence diagnostic is preferred for multi-dimensional applications. Similar to ESS, handling high dimensions and the random variables with zero empirical variance ($W = 0$) is challenging using existing methods [8, 80]. The original Gelman-Rubin's $\hat{R}$ metric has no definition at $W = 0$. Considering each dimension as a separate random variable (RV), we propose an extended procedure (shown in Figure 4) to consider a random variable converged when $B = 0$ and $W = 0$, which indicates all samples are the same value from different iterations and MCMC runs. A random variable is not considered converged when $B > 0$ and $W = 0$, which indicates samples are the same value within MCMC runs, but different across MCMC runs. We propose *convergence percentage*, the percentage of converged univariate random variables, as our new metric. Algorithm 3 outlines the procedure of computing convergence percentage.

[2] An MCMC instance refers to the same input data, model and configuration parameters.

Pillar Insight. Low convergence percentage indicates that more iterations are needed for the model to converge. If a hardware accelerator takes substantially more iterations to converge than the software, the additional iterations may reduce its effective speedup.

Algorithm 3: Convergence percentage

```
Input: trace of multidimensional samples X from m MCMC
       runs, from software or hardware implementations.
num_converged_rvs = 0, num_rvs = 0
for x (trace of each univariable RV) in X do
    if (R̂(x)< 1.1) or (B = 0 and W = 0) then
        num_converged_rvs++
    end
    num_rvs++
end
convergence_percentage =
 num_converged_rvs/num_rvs*100 (in %)
Output: convergence_percentage
```

3.3 Pillar 3: Goodness of Fit

Understanding the "goodness of fit", the difference between end-point results produced by the software and by the hardware accelerator, is important to evaluate the overall quality of the hardware accelerator in absence of ground-truth data. We provide two "goodness of fit" approaches: 1) using application-specific data to measure how good the hardware results fit a reference software result; 2) using a distribution divergence measurement to evaluate all possible data inputs and provide the worst-case divergence.

3.3.1 Application Data Analysis. We are interested in how close or different the results are between the software and hardware. Popular quantitative metrics for "goodness of fit" include Root Mean Squared Error (RMSE) and coefficient of determination (R^2). We choose RMSE given the value of R^2 can be misleading by the small variance of the software results. RMSE measures the root of average squared difference between the result from a hardware MCMC run and a reference software run, ranging from 0 to infinity where lower is better. Due to the stochastic nature of MCMC methods, each MCMC run can have different end-point results for either software or hardware. To account for this variation, we compute RMSE for both hardware and software with respect to a reference software result from multiple MCMC runs. The reference software result is obtained using the mode of multiple software runs to minimize the result variation in a single software reference run.

3.3.2 Data-independent Analysis. Recall the step-1 of sampling is computing the probability distribution to sample from. Hardware approximations in this step introduce divergence from the distribution obtained from FP64-software. Quantifying the distribution divergence of hardware from software provides 1) insights on why the results are good (or bad), 2) how the hardware may perform on unobserved data, and 3) the worst-case divergence with arbitrary input data.

One popular divergence measurement is Kullback-Leibler (KL) divergence. Given the same input data, model, configuration parameters, and states of neighbors, the distribution of a given random variable is computed as P_{sw} from FP64-software and P_{hw} from a hardware implementation. KL divergence (D_{KL}) from P_{hw} to P_{sw} is defined in Equation 10. χ is a collection of all possible outcomes of the random variable and i is a possible outcome.

$$D_{KL}(P_{sw}||P_{hw}) = \sum_{i \in \chi} p_{sw}(i) \log \frac{p_{sw}(i)}{p_{hw}(i)} \tag{10}$$

One major drawback of KL-divergence is it goes to infinity when any entry of $P_{hw}(i)$ is zero while $P_{sw}(i)$ is non-zero, which is likely to happen under the hardware technique of truncating small probabilities to zero, and thus cannot be directly applied to our study. Therefore, we choose Jensen-Shannon Divergence (JSD) as our divergence measurement [50], shown in Equation 11. JSD is defined based on KL-devergence, where $M = (1/2)P_{sw} + (1/2)P_{hw}$. Note that KL-divergence is asymmetric but JSD is symmetric: $D_{JS}(P_{sw}||P_{hw}) = D_{JS}(P_{hw}||P_{sw})$. A lower JSD is preferable, showing distributions of a random variable computed from FP64-software and hardware implementations are close.

$$D_{JS}(P_{sw}||P_{hw}) = \frac{1}{2}D_{KL}(P_{sw}||M) + \frac{1}{2}D_{KL}(P_{hw}||M) \tag{11}$$

Evaluating JSD on arbitrary data inputs for a random variable with many possible labels, such as in stereo vision, is challenging in both analytical and empirical approaches given the complicated mathematical representation and the large parameter space. In this work, we evaluate the JSD in binary label cases, such as in foreground-background image segmentation.

Pillar Insight. Substantially worse RMSE or JSD results for a hardware accelerator means it is likely producing low-quality application end-point results and more iterations or model/hardware design changes may be required.

4 ANALYZING EXISTING HARDWARE

We apply the three pillars of statistical robustness on an existing hardware implementation, the Stochastic Processing Unit (SPU) described in Section 2.4.

4.1 Methodology

In this work, we consider a single SPU as it is sufficient to explore the statistical robustness questions. Development of an accelerator prototype with multiple SPUs is beyond the scope of this work. We primarily utilize MATLAB for both FP64-software and a functionally equivalent SPU simulator. Importantly, we also have SPU implementations in Verilog, Chisel, and HLS all with verified results.

We choose stereo vision and motion estimation as our test applications. Motion estimation infers the motion vectors of image pixels in a frame of a video with respect to its next sequential frame. The concept of applying MRF Gibbs Sampling on motion estimation is similar to stereo vision as described in Section 2.2, except the output label is a 2D motion vector of a pixel, indicating where the pixel moves to in the next frame. Each disparity per pixel in stereo vision is treated as a random variable. Each 2D motion vector per pixel in motion estimation is considered as two random variables x and y. We pick three datasets from Middlebury [2, 70] for each application. We use FP64 runs to find the application parameters (e.g., α and β). Motion estimation has one set of parameters for all datasets, and stereo vision has two sets for all datasets. Some parameters

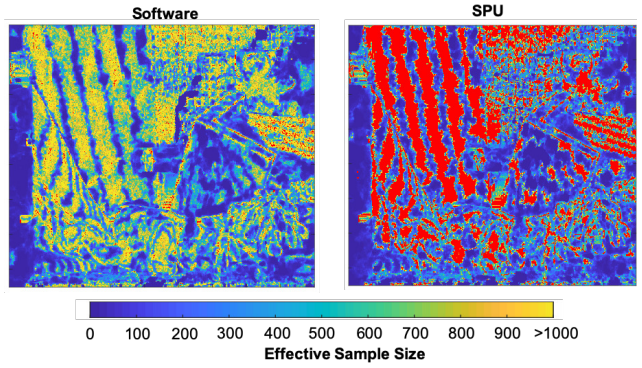


Figure 5: ESS per random variable in stereo vision *teddy*. Red regions correspond to zero variance.

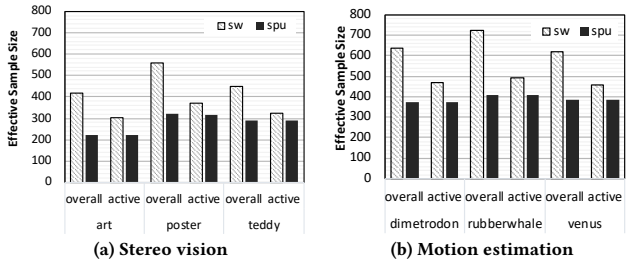


Figure 6: Mean overall and active ESS (higher is better).

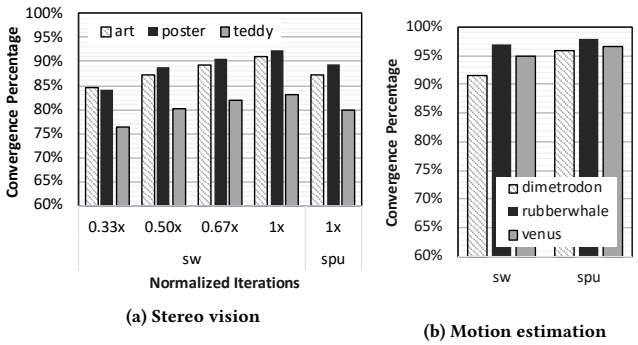


Figure 7: Convergence percentage (higher is better) results. For stereo vision we run software with 0.33×, 0.5×, 0.67×, and 1× the number of SPU iterations, while motion estimation runs the same number of iterations for both.

can be optimized in a training process, which is beyond the scope of this paper. We also considered, but omit, image segmentation since it converges too fast (30 iterations for simulated annealing) to produce meaningful statistical measurements.

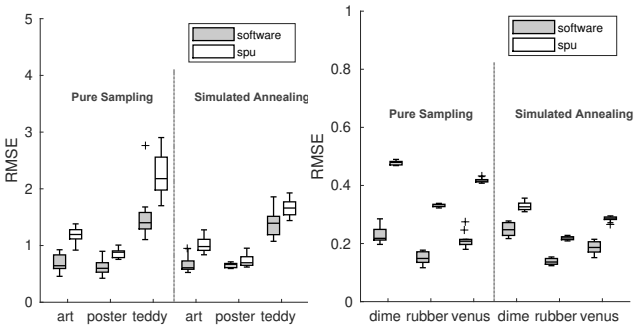
Recall the SPU supports two operating modes: pure sampling that produces the full estimated distribution (sampling), and optimization using simulated-annealing (optimization) that converges quickly to an exact result. For optimization, measuring Effective Sample Size (ESS) and Gelman-Rubin's $\hat{R}$ is not conceptually meaningful, we evaluate sampling quality and convergence diagnostic for sampling only and goodness of fit for both modes. Parameter settings for each dataset are the same in sampling as in optimization, except for a different, fixed temperature. Our empirical results show that all datasets converge after 1,000 iterations for optimization and 3,000 for sampling, except for *poster* in stereo vision which takes only 500 and 1,500 iterations, respectively.

4.2 Results Analysis

4.2.1 Sampling Quality. We analyze ESS on SPU compared with the FP64-software by collecting the last 1,000 iterations of MCMC runs in the two applications. We evaluate the ESS per random variable and report the arithmetic mean. Figure 5 shows an example ESS per random variable in stereo vision *teddy* dataset. Red regions indicate the random variables with zero variance, and thus ESS cannot be calculated. Due to truncating small probabilities to zero, more random variables in the SPU have zero variance than in the software. We consider a random variable with zero variance inactive. The percentage of inactive random variables with respect to the total (a.k.a. inactive percentage) in three stereo vision datasets are 26.9% for *art*, 44.6% for *poster*, and 26.2% for *teddy* in the SPU, compared with 0.3% for *art*, 4.1% for *poster*, and 1.4% for *teddy* in the software. Motion estimation exhibits similar inactive percentages. Zero variance means the probability of a possible label is large enough that all random samples pick the same label empirically, which can indicate convergence. The variance of corresponding inactive random variables in FP64-software is consistently small, indicating the random variable is likely to consistently pick the same label as well—a concentrated distribution. Therefore, a high inactive percentage does not necessarily imply bad result quality.

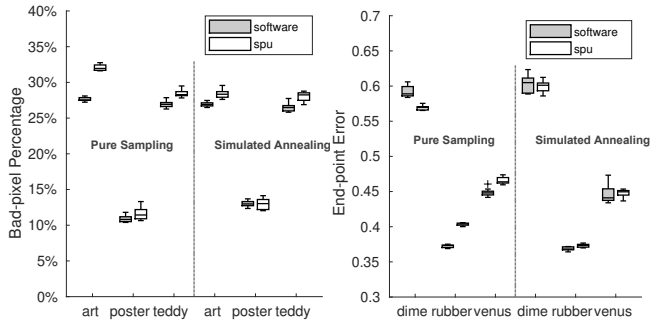
Figure 6 shows the ESS arithmetic mean for a single MCMC run per dataset. We verify that different runs have small ESS differences (< 6 in stereo vision). The mean *overall ESS* eliminates the random variables with zero variance in the software and hardware, respectively. Figure 5 reveals that the inactive regions in the SPU (red) correspond to the regions with high ESS in FP64-software due to small but non-zero variance (yellow), and thus overall ESS is biased toward the software. Therefore, we also report the mean *active ESS* which only includes the regions with non-zero variance in both FP64-software and the SPU, where ESS is more meaningful. As a consequence, the active ESS eliminates the regions with small variance in the software, which can potentially benefit the SPU. The importance of these small variance needs to be evaluated and we are actively looking for methods to account for these regions. The software has 1.1-1.4× higher active ESS than the SPU in stereo vision and around 1.2× in motion estimation. This implies the SPU needs to run 1.1-1.4× iterations to reach the same active ESS as the software. These extra iterations will reduce the speedups (2.8-5.5× and up to 84×) reported in previous work [83, 87].

4.2.2 Convergence Diagnostic. We evaluate the convergence diagnostic of the SPU using the proposed convergence percentage metric. Each convergence percentage value is collected from 10 runs per dataset. Each run forfeits the first half of iterations as the



(a) Stereo vision (b) Motion estimation

Figure 8: Root Mean Squared Error (lower is better). Scales are different in (a) and (b) due to application differences.



(a) Stereo vision (b) Motion estimation

Figure 9: Application end-point result quality (lower is better).

burn-in period and only keeps the second half, as proposed by previous work [8]. Recall a random variable is considered converged if $\hat{R} < 1.1$ or both within-chain variance W and between-chain variance B are zero. Figure 7 shows the results. The number of iterations is normalized with respect to SPU runs in stereo vision (0.33×, 0.5×, 0.67×, and 1×) and are the same in motion estimation. Overall, convergence percentage is high in both the software and the SPU: more than 80% of random variables in stereo vision and more than 90% in motion estimation. More than 99.5% of random variables with $W = 0$ in the SPU are converged. In stereo vision, the SPU reaches the same or better convergence percentage than the software with 2× iterations. This indicates the SPU needs to be at least 2× faster in order to have a better overall performance in this application in terms of convergence percentage. The SPU has higher convergence percentages than the software in motion estimation, indicating the SPU converges faster in this application. Note that converging to a distribution faster does not necessarily lead to a better end-point result. The goodness of fit should be evaluated.

4.2.3 Goodness of Fit. Figure 8 shows the RMSE box plots of 10 MCMC runs per dataset compared with a reference result obtained by the mode of 10 software runs per dataset. Solid boxes show the range from 25th to 75th percentile with the medians of data as the horizontal lines inside. The whiskers include the range of data that are not considered as outliers. We use 1.5× interquartile range as the rule to decide outliers, shown as pluses. The whiskers of the software and the SPU overlap in all stereo vision benchmarks (barely for *teddy* sampling mode), suggesting close results. RMSE results in motion estimation are visually different in Figure 8b. However, these differences are small considering the small scale of y-axis. The software and the SPU produce closer results in simulated annealing optimization mode.

Figure 9 shows the end-point result quality using ground-truth data and application metrics. Most whiskers of the software and the SPU overlap except for *art* in stereo vision and *rubberwhale* in motion estimation, both of which are in sampling mode. In optimization mode, the whiskers of the software and the SPU overlap, indicating the difference in end-point result quality is very small. This is consistent with the single-run results in Table 1. Note that no obvious differences between FP64-software and the SPU are visually observable in the stereo vision disparity maps and the motion estimation flow maps.

It seems intuitive to assume that FP64-software should produce no worse results than the hardware with limited precision, truncation, and a simplified RNG. We find this assumption holds in most but not all cases. We observe that in sampling mode of *dimetrodon*, the SPU has consistently lower end-point result error (Figure 9b) but higher RMSE (Figure 8b) than in FP64-software. To better understand this result, we examine per-pixel differences of end-point error between the software reference and the SPU, as shown in Figure 10. Blue regions correspond to lower end-point error in the SPU and yellow to lower end-point error in the software. The figure suggests the software and the SPU have strengths in different regions, which potentially leads to a high RMSE compared to the software reference.

This result indicates two insights: 1) software with higher precision does not necessarily produce better application end-point result quality, and 2) a higher RMSE compared to software does not always indicate worse application end-point result quality. Although bad pixel-percentage results are consistent with RMSE in stereo vision, the general link between the goodness of fit measure and the application end-point result quality needs to be further explored. *This confirms that collectively applying all three pillars, not just end-point result, is necessary to evaluate correctness.*

We analyze the Jensen-Shannon Divergence of SPU relative to the software with FP64 probability representation. Our goal is to provide insights on why hardware exhibits good or bad application end-point results and how it may perform with arbitrary input data. We assume each random variable has a binary distribution in this analysis. By sweeping a wide range of possible energy inputs $E(i)$ (refer to Equation 1) from 0 to 255 in integer, corresponding to arbitrary data inputs, Figure 11 plots JSD for two temperatures (1 and 10) and two different SPU microarchitectures: 1) the SPU described in Section 2.4 and 2) an early design [83]—1st-gen RSU-G—that was shown to lack sufficient precision and dynamic range [87]. These results clearly show the problems with the 1st-gen RSU-G. The more recent SPU has negligible JSDs in most energy inputs (blue regions), whereas the 1st-gen RSU-G has high JSD (>0.2, yellow) for

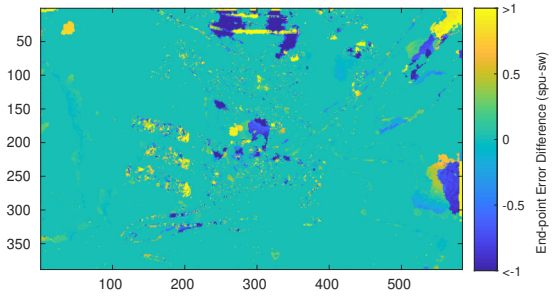


Figure 10: *Dimetrodon* end-point error difference ($spu-sw$) at pixel level. End-point error: 0.581 (software) vs. 0.567 (SPU).

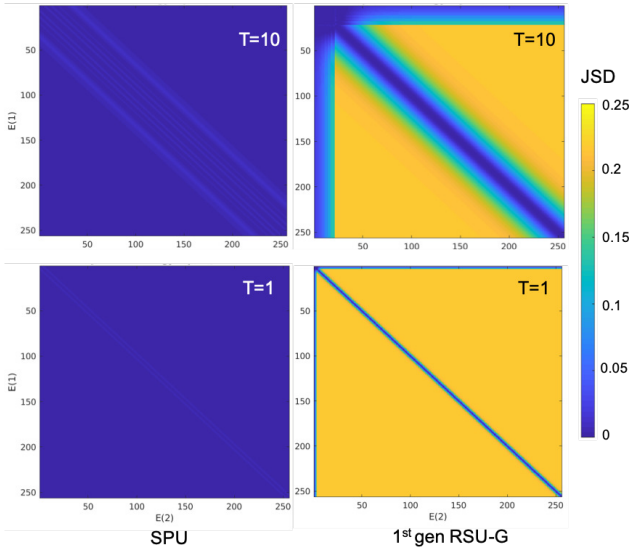


Figure 11: Jensen-Shannon Divergence comparison between designs: SPU vs. 1st-gen RSU-G [83].

many inputs and becomes worse when the temperature decreases, which explains the poor application result quality. A key difference between these two designs is dynamic scaling for energy values in the SPU, which is not present in the 1st-gen RSU-G.

5 DESIGN SPACE EXPLORATION: A CASE STUDY

The previous section shows that architectural optimizations might have a negative influence on statistical robustness, even while producing comparable end-point results to FP64-software. The question is *can we achieve desirable end-point result quality and statistical robustness without the commensurate overhead of FP64?* To answer this question, we use the three pillars to explore the SPU design space.

5.1 Design Trade-offs

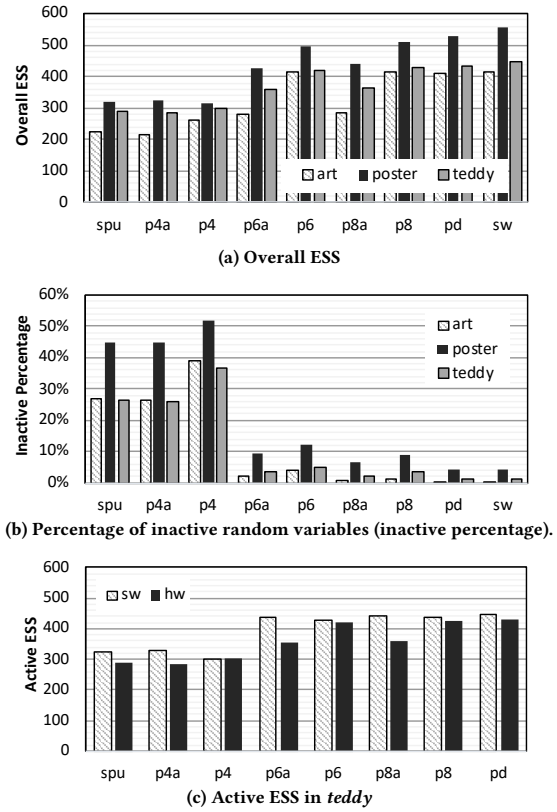
The SPU pipeline (Figure 3) has several design parameters related to bit precision that potentially influence statistical robustness, including energy $E(i)$ and $E_s(i)$, scaled and truncated probability $p_{tr}(i)$, and RNG output bits. We fix energy $E(i)$ and $E_s(i)$ at 8 bits based on previous work [57, 87]. The number of bits in $p_{tr}(i)$ considerably influences the size of the energy-to-probability converter and the discrete sampler. We evaluate three design points with 4-bit, 6-bit, and 8-bit $p_{tr}(i)$s. The influence of RNG output bits is small compared to $p_{tr}(i)$ and we find a 19-bit LFSR with 12-bit RNG outputs does not reduce the statistical robustness or result quality across all design points, although the period is notably short compared with RNGs used in other accelerators [9, 47, 57]. Understanding the influence of RNGs on the probabilistic algorithms is very challenging and needs continual efforts [13, 14, 71, 77].

The SPU truncates all $p_{tr}(i)$s to the nearest 2^n values, called 2^n approximation [87], enabling efficient energy-to-probability conversion by comparing the boundaries of energy values. Without 2^n approximation, a double-buffered 256-entry LUT is required to store the $p_{tr}(i)$ values to achieve a stall-free design. We evaluate the statistical robustness of each scaled probability design point with and without 2^n approximation. The above design parameters generally do not directly influence the SPU per-iteration execution time assuming the same interface at the same clock frequency. However, a design with lower precision may take more iterations to converge. On the other hand, higher precision requires more area and power affecting the number of SPU units in systems with a limited area/power budget. Detailed system-level architecture investigations are beyond the scope of this paper.

5.2 Evaluating Design Parameters

Figures 12-19 show our design space results. We explain stereo vision results in detail and highlight motion estimation results where needed since both applications show consistent results. Starting from the current SPU design ("spu"), we analyze the statistical robustness by gradually increasing the precision: 1) replace the 19-bit LFSR sampler with an FP64 Mersenne Twister sampler while keeping the front-end pipeline unchanged ("p4a"); 2) increase the bit width of $p_{tr}(i)$ to 6, 8-bit ("p6a" and "p8a"), with 2^n approximation; 3) remove 2^n approximation ("p4", "p6", and "p8"); and 4) keep front-end pipeline up to the scaled energy ($E_s(i)$) output unchanged, but has an FP64 back-end for probability conversion and discrete sampling ("pd").

5.2.1 Sampling Quality. Figure 12a shows overall ESS, which omits random variables with zero variance for each design, respectively. Recall this metric can create bias that benefits FP64-software for variables with small but non-zero variance. Overall ESSs increase when more bits are added, partly as a result of fewer random variables with zero variance. Recall the SPU truncates small scaled probabilities $p_{tr}(i) < 1$ to zero. Adding more bits keeps more possible labels with small probabilities available to be sampled. Figure 12b indicates inactive percentage drops significantly when increasing $p_{tr}(i)$ bit size from 4 to 6. Interestingly, 2^n approximation helps reduce the inactive percentage under the same bit precision, but decreases ESS for 6-bit and 8-bit designs. Figure 12c shows the active ESS for the *teddy* dataset. Recall active ESS masks out the random variables inactive in either FP64-software or the SPU. With 2^n approximation, increasing bit precision does not close the gap in active ESS with the software. Without 2^n approximation, 6 or 8-bit $p_{tr}(i)$ have comparable overall and active ESS to software. As expected, increasing bit precision decreases the difference between



(a) Overall ESS

(b) Percentage of inactive random variables (inactive percentage).

(c) Active ESS in *teddy*

Figure 12: Stereo vision sampling quality in the design points.

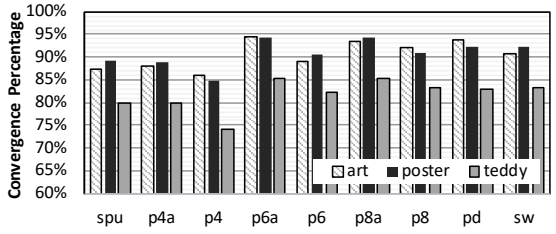
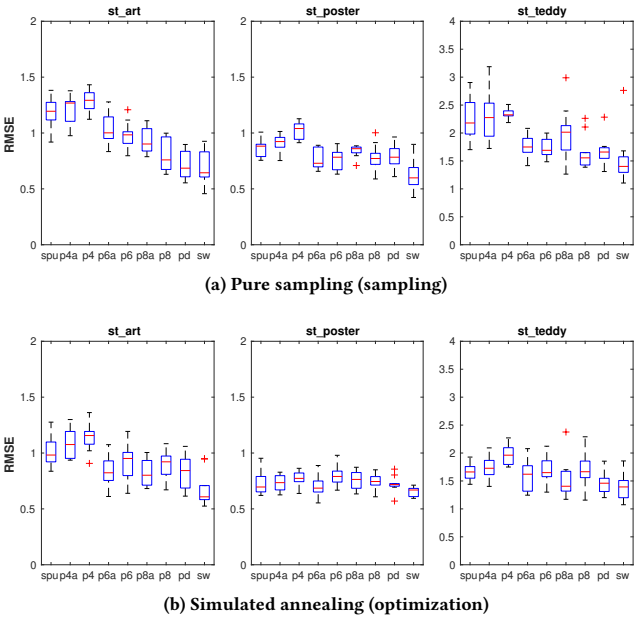


Figure 13: Stereo vision convergence percentage in the design points.

overall and active ESS due to fewer inactive variables. Results for motion estimation are similar, shown in Figure 16.

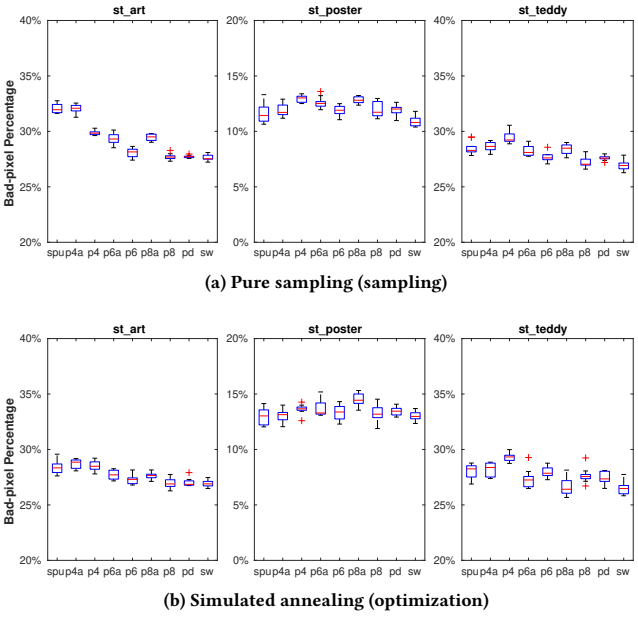
5.2.2 Convergence Diagnostic. Figure 13 shows the convergence percentage increases with the increasing bit precision. In contrast to ESS, 2^n approximation improves the convergence percentage under the same bit precision. Hardware with 6-bit and 8-bit $p_{tr}(i)$ with and without 2^n approximation produces comparable convergence percentage to FP64-software. All designs except "p4" produce the same or higher convergence percentage compared with FP64-software for motion estimation shown in Figure 17. All values of convergence percentage are high (> 90%) in motion estimation.



(a) Pure sampling (sampling)

(b) Simulated annealing (optimization)

Figure 14: Stereo vision RMSE in the design points.

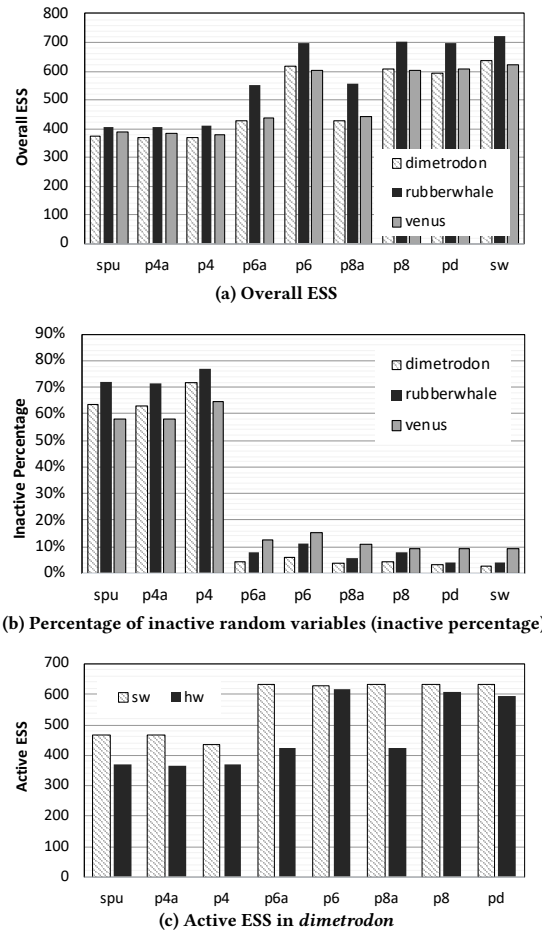


(a) Pure sampling (sampling)

(b) Simulated annealing (optimization)

Figure 15: Stereo vision application end-point result quality in the design points.

5.2.3 Goodness of Fit. Figure 14 shows RMSE results compared with software reference results. Observable lower RMSEs can be found in stereo vision *art* when increasing the bit precision from 4 to 6. Differences of RMSEs are hard to notice when further increasing the precision given whiskers largely overlap in most datasets.



(a) Overall ESS

(b) Percentage of inactive random variables (inactive percentage).

(c) Active ESS in *dimetrodon*

Figure 16: Motion estimation sampling quality in the design points.

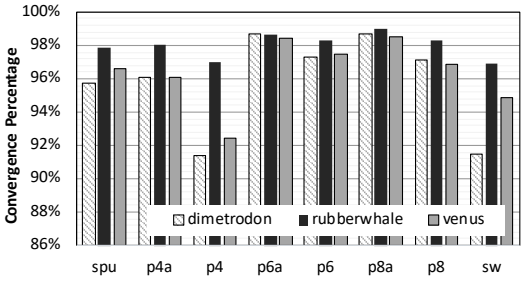
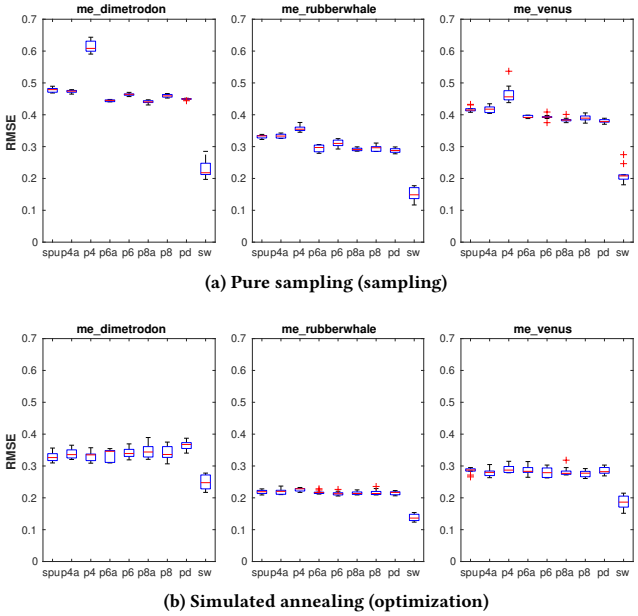


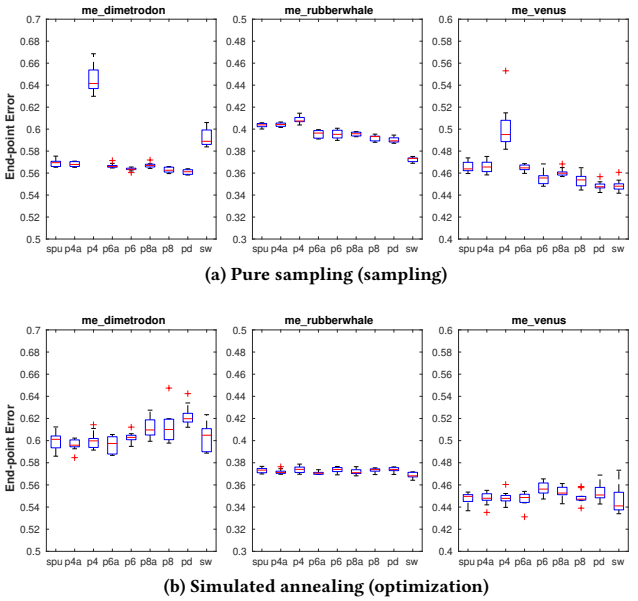
Figure 17: Motion estimation convergence percentage in the design points.



(a) Pure sampling (sampling)

(b) Simulated annealing (optimization)

Figure 18: Motion estimation RMSE in the design points.



(a) Pure sampling (sampling)

(b) Simulated annealing (optimization)

Figure 19: Motion estimation application end-point result quality in the design points.

Application end-point results in Figure 15 exhibit the same trends. All designs produce comparable result quality to the software in simulated annealing (optimization), consistent with Table 1. We highlight the following results for motion estimation (Figures 18 and 19): 1) the design parameters have a negligible influence on application end-point result quality (end-point error) except "p4" in a couple of cases, which performs observably worse; 2) all designs except "p4" produce better end-point error than the software for *dimetrodon* with sampling; 3) all designs produce slightly worse end-point error than the software for *rubberwhale* with sampling; and 4) gaps exist between the software and all hardware designs

Table 2: Resource usage and performance of various SPU implementations on Arria 10 FPGA.

Parameter	Verilog	HLS-int	HLS-fp
Frequency (MHz)	374	369	320
ALMs	321	1,189	4,407
Registers	680	2,551	7,932
Memory Bits	1,472	10,688	49,536
DSPs	4	10	25
Initiation Interval (Cycles)	1	1	3

Table 3: Area (μm^2) and Power (mW) Analysis in ASIC.

Design	Area	Power	Design	Area	Power
spu	1957	2.17	p4	2112	2.21
p6a	2134	2.31	p6	2356	2.38
p8a	2309	2.46	p8	2599	2.54

including "pd" for RMSE, but not for end-point error. These results confirm the importance of using all three pillars. Overall, we summarize 1) optimization is more robust than sampling at producing good result quality across various designs, and 2) increasing the scaled probability to 6 bits produces comparable goodness of fit and end-point results to FP64-software.

5.3 Resource Usage

5.3.1 FPGA. We evaluate three different implementations of the SPU on an Intel Arria 10 FPGA [40]: 1) an optimized hand-written Verilog implementation with 4-bit scaled probability, 2) a High-Level Synthesis (HLS) implementation (HLS-int) that matches the hand-written Verilog (but using HLS basic integer data-types), and 3) an HLS implementation with a 32-bit floating-point (FP32) back-end after energy computation (HLS-fp), eliminating the energy scaling stage. HLS-fp is developed in order to assess the option of directly using FP32 representation inside the SPU for probability conversion and sampling. Table 2 shows the synthesis results. HLS-int is close to the Verilog implementation in terms of performance (frequency and initiation interval), but consumes more resources. The resource usage of HLS-int can be further decreased by using reduced-precision integers. HLS-fp consumes 13.7× ALMs, 33.7× memory, 6.3× DSPs compared to Verilog and most importantly performs remarkably worse due to its lower frequency and throughput (initiation interval) caused by the FP addition [39]. Clearly, naively implementing the SPU in FP32 consumes too many resources and significantly reduces the performance benefits. A human-designed architecture is needed to improve efficiency.

5.3.2 ASIC. We estimate the ASIC area/power for various design points. Circuit elements written in Chisel are synthesized in a predictive 15nm library [59] using Synopsys Design Compiler. Memory elements (FIFOs and LUTs) are estimated using Cacti 7 [3] in 22nm technology, the smallest supported technology. The designs are verified using stereo vision *art*. Table 3 summarizes the results. Total area/power numbers are the sum of 15nm circuitry and 22nm memory elements. Power is estimated at 1GHz. Since Cacti requires widths in multiples of bytes, we estimate a double-buffered 2×256-byte LUT (537 μm^2 and 0.32 mW) and a 64-byte FIFO (215 μm^2 and 0.18 mW) with an 8-bit port, and linearly scale them to target widths of 4 and 6 bits. All designs can run up to 3.3GHz, bounded by the SPU energy computation stage. Increasing the SPU $p_{tr}(i)$ from 4-bit to 6-bit precision while keeping the 2^n approximation ("p6a") incurs 1.09× area and 1.07× power overheads, but has considerably better statistical robustness. Removing 2^n approximation ("p6") adds double-buffered LUTs for energy-to-probability conversion, thus incurs 1.20× area and 1.10× power overheads. Despite a 10% difference in area, we advocate the 6-bit designs without 2^n approximation in an ASIC for better sampling quality if area is not a major concern. The benefit from further increasing the bit-precision is marginal based on the previous analysis.

6 DISCUSSION

Our work takes an important first step to introduce the architecture community to widely used evaluation methods from the statistics community. Historically, the popular approach to evaluating design correctness of an accelerator is to simply compare end-point results against a software baseline considered accurate using standard input data sets and metrics. We argue this is necessary but not sufficient to claim correctness of probabilistic accelerators given domain experts in statistics are interested in the full distribution of the possible results, including end-point results and quantified uncertainty (statistical robustness). A degradation in statistical robustness may require more algorithm iterations and thus offset the improved efficiency provided by hardware/software optimization. Our proposed framework—the three pillars of statistical robustness—includes techniques commonly used by domain experts to quantify uncertainty and make statistical guarantees regarding full distributions. In absence of ground-truth, the three pillars provide confidence about end-point results and uncertainty. Good statistical robustness is likely to achieve good end-point results. Nevertheless, we advocate assessing both end-point results and statistical robustness, when possible, for correctness. Note that the proposed pillars can be used for software implementations that exploit recently introduced data types, such as BFloat16 [82] or Microsoft Floating Point [67]. We utilize the pillars in the context of hardware since the trade-off between robustness and performance/power/area is critical and has been previously overlooked.

We discuss statistical robustness starting with MCMC accelerators where domain experts have clear agreement on the appropriate evaluation metrics. The selected pillars and metrics are well-accepted by Bayesian statisticians, although we do not claim completeness we believe they comprehensively cover key aspects of statistical robustness. The analysis of other metrics and methods (e.g., MCMC standard error [20]) might help identify limitations of selected metrics. The challenges of directly applying existing metrics motivate us to propose modified processes and a new metric (convergence percentage) for reporting scalar measures for sampling quality and convergence diagnostic. Our proposals are conceptually straightforward, but could benefit from domain experts developing metrics with stronger theoretical foundations. Meanwhile, the adequateness of rule-of-thumb $\hat{R} < 1.1$ to determine

convergence is under debate [80]. Additionally, defining "good enough" statistical robustness is a valuable future work.

Our work uses a representative MCMC accelerator and two computer vision applications as a case study. Applying the three pillars to other MCMC accelerators, applications, and models remains future work. Our proposed processes and metrics apply to other MCMC accelerators and applications, especially for those when directly applying existing methods is difficult due to high dimensionality and random variables with zero variance. Applications with continuous random variables and low dimensionality may utilize the unmodified metrics within each pillar. The effects of some hardware approximations are unknown for applications that require information from variables with very low variance, such as rare event simulation.

We argue that bringing statistical robustness into the architecture design process applies to all types of accelerators, with the evolution/refinement of pillars/metrics. The key to choosing pillars/metrics for broader use is to appropriately address uncertainty. The proposed pillars or metrics may be used for other probabilistic algorithms. For example, both ESS and $\hat{R}$ are used for evaluating performance and robustness of Hamiltonian Monte Carlo [7]. Convergence diagnostic *pflug* is used for evaluating the process of Stochastic Gradient Descent [11, 64]. Diagnostics are available for Variational Inference, such as Pareto Smoothed Importance Sampling (PSIS, or $\hat{k}$) [81] and Variational Simulation-Based Calibration (VSBC) [86]. These metrics may be useful in our framework to indicate robustness when evaluating the corresponding accelerators.

We believe the idea of evaluating statistical robustness is also necessary for Deep Neural Networks (DNN) if full distributions are of interest, which may be achievable if using appropriate uncertainty metrics. Compared with MCMC, metrics for quantifying uncertainty in DNNs are various and rapidly evolving. Krishnan and Tickoo [48] summarize several metrics for classification tasks, including predictive entropy [72], variation ratio [21], and mutual information [72]. Recent work proposes methods to extract aleatory and/or epistemic uncertainties, such as those proposed by Kendall and Gal [43], Kwon et al. [49], Postels et al. [65], Monte-Carlo dropout [22], Prior Networks [56], etc. It is interesting to see how these metrics could be adapted into the process of designing robust DNN accelerators.

7 RELATED WORK

Section 2.3 discussed various specialized accelerators for probabilistic algorithms. Other accelerators exist for deterministic Bayesian Inference [38, 51]. A benchmark for Bayesian Inference models is proposed for performance evaluation [84]. Previous work addresses some statistical metrics for MCMC accelerators. Mansinghka and Jonas [57] evaluate data input precision using KL-divergence and QQ plots. Liu et al. [53] argue using ESS/second as a performance metric for MCMC samplers. Mingas et al. [61] use both ESS/second and KL-divergence. A recent GPU MCMC method [31] uses a convergence diagnostic to decide stopping point. Multiple goodness of fit statistical tests exist, such as Kolmogorov–Smirnov test (KS-test), Analysis of Variance (ANOVA), a kernel two-sample test [29], a goodness-of-fit test based on Stochastic Rank Statistic (SRS) [68], etc. These metrics all belong to one of three pillars proposed in this work and we argue all three pillars are needed to fully characterize the statistical robustness of an MCMC accelerator. Theoretical studies provide error bounds for MCMC with algorithmic approximation techniques given mathematical assumptions [23, 41]. Studies (e.g., Darulova [18]) and tools (e.g. Gappa++ [52]) are available for evaluating quantization error, which are important but orthogonal to statistical robustness. Methods are proposed to make statistical guarantees on end-point results for approximate computing [55, 63], which are conceivably useful to find acceptable end-point quality loss and to evaluate goodness-of-fit. Analytical and empirical studies have been done on evaluating limited precision in neural networks [15, 30, 36, 69].

8 CONCLUSION

Probabilistic computing is an important branch in statistical machine learning with the advantage of interpretability and generality. Many specialized architectures using approximation techniques have been proposed to address sampling inefficiency of probabilistic algorithms. These domain-specific accelerators should provide correct execution of the original algorithms. Current evaluation methodologies that focus only on end-point result quality are not adequate for evaluating probabilistic algorithms, since the correctness defined by domain experts includes both end-point results and statistical robustness. Therefore, we claim *a probabilistic architecture should provide some measure (or guarantee) of statistical robustness.* This work takes a first step toward defining metrics and a framework for evaluating correctness of probabilistic accelerators beyond application end-point result quality.

We propose three pillars of statistical robustness: 1) sampling quality, 2) convergence diagnostic, and 3) goodness of fit. The pillars are built on quantitative metrics with the necessary modifications to account for high-dimensionality and zero empirical variance that occurs in practical cases. We apply the three pillars to an existing hardware accelerator and surface design issues that cannot be revealed by only using application end-point result quality. We show how the three pillars can guide design space exploration and achieve considerable improvements in statistical robustness by slightly increasing bit precision. This work takes an important step in raising awareness that correctness for probabilistic accelerators is more than application end-point accuracy.

ACKNOWLEDGMENTS

This project is supported in part by Intel, the Semiconductor Research Corporation and the National Science Foundation (CNS-1616947). We thank Rong Ge and Cheng Lyu.

REFERENCES

[1] Tarek Ould Bachir, Mohamad Sawan, and Jean-Jules Brault. 2008. A new hardware architecture for sampling the exponential distribution. In *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*. IEEE, 001393–001396. https://doi.org/10.1109/CCECE.2008.4564770

[2] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. 2011. A Database and Evaluation Methodology for Optical Flow. *International Journal of Computer Vision* 92, 1 (01 Mar 2011), 1–31. https://doi.org/10.1007/s11263-010-0390-2

[3] Rajeev Balasubramonian, Andrew B. Kahng, Naveen Muralimanohar, Ali Shafiee, and Vaishnav Srinivas. 2017. CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Trans. Archit. Code Optim.* 14, 2, Article 14 (June 2017), 25 pages. https://doi.org/10.1145/3085572

[4] Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. 2019. AcMC 2 : Accelerating Markov Chain Monte Carlo Algorithms for Probabilistic Models. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems* (Providence, RI, USA) *(ASPLOS '19)*. ACM, New York, NY, USA, 515–528. https://doi.org/10.1145/3297858.3304019

[5] Aubrey Barnard. 2019. *Causal Discovery of Adverse Drug Events in Observational Data.* Ph.D. Dissertation. University of Wisconsin–Madison.

[6] Stephen T. Barnard. 1989. Stochastic stereo matching over scale. *International Journal of Computer Vision* 3, 1 (01 May 1989), 17–32. https://doi.org/10.1007/BF00054836

[7] Michael Betancourt. 2017. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint* (2017). arXiv:1701.02434

[8] Stephen P. Brooks and Andrew Gelman. 1998. General Methods for Monitoring Convergence of Iterative Simulations. *Journal of Computational and Graphical Statistics* 7, 4 (1998), 434–455. https://doi.org/10.1080/10618600.1998.10474787

[9] Ruizhe Cai, Ao Ren, Ning Liu, Caiwen Ding, Luhao Wang, Xuehai Qian, Massoud Pedram, and Yanzhi Wang. 2018. VIBNN: Hardware Acceleration of Bayesian Neural Networks *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 476–488. https://doi.org/10.1145/3173162.3173212

[10] Lakshmi N. Chakrapani, Bilge E.S. Akgul, Suresh Cheemalavagu, Pinar Korkmaz, Krishna V. Palem, and Balasubramanian Seshasayee. 2006. Ultra-Efficient (Embedded) SOC Architectures based on Probabilistic CMOS (PCMOS) Technology. In *Proceedings of the Design Automation Test in Europe Conference*, Vol. 1. 1–6. https://doi.org/10.1109/DATE.2006.243978

[11] Jerry Chee and Panos Toulis. 2018. Convergence diagnostics for stochastic gradient descent with constant learning rate. In *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 84)*, Amos Storkey and Fernando Perez-Cruz (Eds.). PMLR, 1476–1485. http://proceedings.mlr.press/v84/chee18a.html

[12] Wenjun Cheng, Luyao Ma, Tiejun Yang, Jiali Liang, and Yan Zhang. 2016. Joint lung CT image segmentation: a hierarchical Bayesian approach. *PloS one* 11, 9 (2016). https://doi.org/10.1371/journal.pone.0162211

[13] Timothy H. Click, Aibing Liu, and George A. Kaminski. 2011. Quality of random number generators significantly affects results of Monte Carlo simulations for organic and biological systems. *Journal of computational chemistry* 32, 3 (2011), 513–524. https://doi.org/10.1002/jcc.21638

[14] Paul D. Coddington. 1994. Analysis of random number generators using Monte Carlo simulation. *International Journal of Modern Physics C* 5, 03 (1994), 547–560. https://doi.org/10.1142/S0129183194000726

[15] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training Deep Neural Networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 3123–3131. http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-weights-during-propagations.pdf

[16] Mary Kathryn Cowles and Bradley P. Carlin. 1996. Markov chain Monte Carlo convergence diagnostics: a comparative review. *J. Amer. Statist. Assoc.* 91, 434 (1996), 883–904. https://doi.org/10.1080/01621459.1996.10476956

[17] Keivan Dabiri, Mehrdad Malekmohammadi, Ali Sheikholeslami, and Hirotaka Tamura. 2020. Replica Exchange MCMC Hardware With Automatic Temperature Selection and Parallel Trial. *IEEE Transactions on Parallel and Distributed Systems* 31, 7 (2020), 1681–1692. https://doi.org/10.1109/TPDS.2020.2972359

[18] Eva Darulova. 2014. *Programming with numerical uncertainties.* Ph.D. Dissertation. EPFL. https://doi.org/10.5075/epfl-thesis-6343

[19] Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd annual meeting on association for computational linguistics*. Association for Computational Linguistics, 363–370. https://doi.org/10.3115/1219840.1219885

[20] James M. Flegal, Murali Haran, and Galin L. Jones. 2008. Markov chain Monte Carlo: Can we trust the third significant figure? *Statist. Sci.* (2008), 250–260. https://doi.org/10.1214/08-STS257

[21] Linton C. Freeman. 1965. *Elementary Applied Statistics: For Students in Behavioral Science.* Wiley.

[22] Yarin Gal and Zoubin Ghahramani. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*. 1050–1059. http://proceedings.mlr.press/v48/gal16.html

[23] Rong Ge, Holden Lee, and Andrej Risteski. 2018. Simulated Tempering Langevin Monte Carlo II: An Improved Proof using Soft Markov Chain Decomposition. *arXiv preprint* (2018). arXiv:1812.00793

[24] Andrew Gelman and Donald B. Rubin. 1992. Inference from iterative simulation using multiple sequences. *Statistical science* 7, 4 (1992), 457–472. https://doi.org/10.1214/ss/1177011136

[25] Stuart Geman and Donald Geman. 1984. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), 721–741. https://doi.org/10.1109/TPAMI.1984.4767596

[26] Sinong Geng, Zhaobin Kuang, Jie Liu, Stephen Wright, and David Page. 2018. Stochastic learning for sparse discrete markov random fields with controlled gradient approximation error. In *Conference on Uncertainty in Artificial Intelligence*, Vol. 2018. NIH Public Access, 156. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6292514

[27] Zoubin Ghahramani. 2015. Probabilistic machine learning and artificial intelligence. *Nature* 521, 7553 (2015), 452–459. https://doi.org/10.1038/nature14541

[28] Lei Gong and James M. Flegal. 2016. A practical sequential stopping rule for high-dimensional Markov chain Monte Carlo. *Journal of Computational and Graphical Statistics* 25, 3 (2016), 684–700. https://doi.org/10.1080/10618600.2015.1044092

[29] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. 2012. A kernel two-sample test. *Journal of Machine Learning Research* 13, Mar (2012), 723–773. https://www.jmlr.org/papers/volume13/gretton12a/gretton12a.pdf

[30] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. 2015. Deep Learning with Limited Numerical Precision. *arXiv preprint* (2015). arXiv:1502.02551

[31] Allan Haldane and Ronald M. Levy. 2020. Mi3-GPU: MCMC-based inverse ising inference on GPUs for protein covariation analysis. *Computer Physics Communications* (2020), 107312. https://doi.org/10.1016/j.cpc.2020.107312

[32] Ghassan Hamra, Richard MacLehose, and David Richardson. 2013. Markov chain Monte Carlo: an introduction for epidemiologists. *International journal of epidemiology* 42, 2 (2013), 627–634. https://doi.org/10.1093/ije/dyt043

[33] Marcel Häselich, Simon Eggert, and Dietrich Paulus. 2012. Parallelized energy minimization for real-time Markov random field terrain classification in natural environments. In *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE, 1823–1828. https://doi.org/10.1109/ROBIO.2012.6491233

[34] Jonathan C. Hedstrom, Chung Him Yuen, Rong-Rong Chen, and Behrouz Farhang-Boroujeny. 2017. Achieving near MAP performance with an excited Markov chain Monte Carlo MIMO detector. *IEEE Transactions on Wireless Communications* 16, 12 (2017), 7718–7732. https://doi.org/10.1109/TWC.2017.2750667

[35] Gregory Herschlag, Han Sung Kang, Justin Luo, Christy Vaughn Graves, Sachet Bangia, Robert Ravier, and Jonathan C. Mattingly. 2018. Quantifying gerrymandering in north carolina. *arXiv preprint* (2018). arXiv:1801.03783

[36] Jordan L. Holi and Jenq-Neng Hwang. 1993. Finite precision error analysis of neural network hardware implementations. *IEEE Trans. Comput.* 42, 3 (1993), 281–290. https://doi.org/10.1109/12.210171

[37] Eyke Hüllermeier and Willem Waegeman. 2019. Aleatoric and Epistemic Uncertainty in Machine Learning: An Introduction to Concepts and Methods. *arXiv preprint* (2019). arXiv:1910.09457

[38] Skand Hurkat and José F Martínez. 2019. VIP: A Versatile Inference Processor. In *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 345–358. https://doi.org/10.1109/HPCA.2019.00049

[39] Intel®. 2019. Floating-Point IP Cores User Guide. https://www.intel.com/content/www/us/en/programmable/documentation/eis1410764818924.html

[40] Intel®. 2019. Intel® Quartus® Prime Software Suite. https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html

[41] James E. Johndrow, Jonathan C. Mattingly, Sayan Mukherjee, and David Dunson. 2015. Optimal approximating Markov chains for Bayesian inference. *arXiv preprint* (2015). arXiv:1508.03387

[42] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal. 1998. Markov chain Monte Carlo in practice: a roundtable discussion. *The American Statistician* 52, 2 (1998), 93–100. https://doi.org/10.2307/2685466

[43] Alex Kendall and Yarin Gal. 2017. What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 5580–5590.

[44] Osama U. Khan and David D. Wentzloff. 2016. Hardware Accelerator for Probabilistic Inference in 65-nm CMOS. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24, 3 (2016), 837–845. https://doi.org/10.1109/TVLSI.2015.2420663

[45] Paresh Kharya. 2020. NVIDIA Blogs: TensorFloat-32 Accelerates AI Training HPC upto 20x. https://blogs.nvidia.com/blog/2020/05/14/tensorfloat-32-precision-format/

[46] Leslie Kish. 1965. *Survey sampling.* New York: John Wiley & Sons.

[47] Glenn G. Ko, Yuji Chai, Rob A. Rutenbar, David Brooks, and Gu-Yeon Wei. 2019. Accelerating Bayesian Inference on Structured Graphs Using Parallel Gibbs Sampling. In *2019 29th International Conference on Field Programmable Logic and Applications (FPL)*. 159–165. https://doi.org/10.1109/FPL.2019.00033

[48] Ranganath Krishnan and Omesh Tickoo. 2020. Improving model calibration with accuracy versus uncertainty optimization. *arXiv preprint* (2020). arXiv:2012.07923

[49] Yongchan Kwon, Joong-Ho Won, Beom Joon Kim, and Myunghee Cho Paik. 2020. Uncertainty quantification using Bayesian neural networks in classification: Application to biomedical image segmentation. *Computational Statistics & Data Analysis* 142 (2020), 106816. https://doi.org/10.1016/j.csda.2019.106816

[50] Jianhua Lin. 1991. Divergence measures based on the Shannon entropy. *IEEE Transactions on Information theory* 37, 1 (1991), 145–151. https://doi.org/10.1109/18.61115

[51] Mingjie Lin, Ilia Lebedev, and John Wawrzynek. 2010. High-throughput Bayesian Computing Machine with Reconfigurable Hardware. In *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (Monterey, California, USA) *(FPGA '10)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/1723112.1723127

[52] Michael D. Linderman, Matthew Ho, David L. Dill, Teresa H. Meng, and Garry P. Nolan. 2010. Towards Program Optimization Through Automated Analysis of Numerical Precision. In *Proceedings of the 8th Annual IEEE/ACM International Symposium on Code Generation and Optimization* (Toronto, Ontario, Canada) *(CGO '10)*. ACM, New York, NY, USA, 230–237. https://doi.org/10.1145/1772954.1772987

[53] Shuanglong Liu, Grigorios Mingas, and Christos-Savvas Bouganis. 2015. An exact MCMC accelerator under custom precision regimes. In *2015 International Conference on Field Programmable Technology (FPT)*. IEEE, 120–127. https://doi.org/10.1109/FPT.2015.7393138

[54] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. 2016. Tabla: A unified template-based framework for accelerating statistical machine learning. In *High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on*. IEEE, 14–26. https://doi.org/10.1109/HPCA.2016.7446050

[55] Divya Mahajan, Amir Yazdanbakhsh, Jongse Park, Bradley Thwaites, and Hadi Esmaeilzadeh. 2016. Towards Statistical Guarantees in Controlling Quality Tradeoffs for Approximate Acceleration. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*. 66–77. https://doi.org/10.1109/ISCA.2016.16

[56] Andrey Malinin and Mark Gales. 2018. Predictive Uncertainty Estimation via Prior Networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems* (Montréal, Canada) *(NIPS'18)*. Curran Associates Inc., Red Hook, NY, USA, 7047–7058.

[57] Vikash Mansinghka and Eric Jonas. 2014. Building fast Bayesian computing machines out of intentionally stochastic, digital parts. *arXiv preprint* (2014). arXiv:1402.4914

[58] Luca Martino, Víctor Elvira, David Luengo, Jukka Corander, and Francisco Louzada. 2016. Orthogonal parallel MCMC methods for sampling and optimization. *Digital Signal Processing* 58 (2016), 64–84. https://doi.org/10.1016/j.dsp.2016.07.013

[59] Mayler Martins, Jody Maick Matos, Renato P. Ribas, André Reis, Guilherme Schlinker, Lucio Rech, and Jens Michelsen. 2015. Open Cell Library in 15Nm FreePDK Technology. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design* (Monterey, California, USA) *(ISPD '15)*. ACM, New York, NY, USA, 171–178. https://doi.org/10.1145/2717764.2717783

[60] Patrick McClure, Nao Rho, John A. Lee, Jakub R. Kaczmarzyk, Charles Y. Zheng, Satrajit S. Ghosh, Dylan M. Nielson, Adam G. Thomas, Peter Bandettini, and Francisco Pereira. 2019. Knowing What You Know in Brain Segmentation Using Bayesian Deep Neural Networks. *Frontiers in Neuroinformatics* 13 (2019), 67. https://doi.org/10.3389/fninf.2019.00067

[61] Grigorios Mingas, Leonardo Bottolo, and Christos-Savvas Bouganis. 2017. Particle MCMC algorithms and architectures for accelerating inference in state-space models. *International Journal of Approximate Reasoning* 83 (2017), 413–433. https://doi.org/10.1016/j.ijar.2016.10.011

[62] Radford M. Neal. 2011. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* 2, 11 (2011), 2. arXiv:1206.1901

[63] Jongse Park, Emmanuel Amaro, Divya Mahajan, Bradley Thwaites, and Hadi Esmaeilzadeh. 2016. AxGames: Towards Crowdsourcing Quality Target Determination in Approximate Computing. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems* (Atlanta, Georgia, USA) *(ASPLOS '16)*. Association for Computing Machinery, New York, NY, USA, 623–636. https://doi.org/10.1145/2872362.2872376

[64] Georg Ch Pflug. 1992. Gradient estimates for the performance of Markov chains and discrete event processes. *Annals of Operations Research* 39, 1 (1992), 173–194. https://doi.org/10.1007/BF02060941

[65] Janis Postels, Francesco Ferroni, Huseyin Coskun, Nassir Navab, and Federico Tombari. 2019. Sampling-free epistemic uncertainty estimation using approximated variance propagation. In *Proceedings of the IEEE International Conference on Computer Vision*. 2931–2940. https://doi.org/10.1109/ICCV.2019.00302

[66] Christian Robert and George Casella. 2013. *Monte Carlo statistical methods*. Springer Science & Business Media. https://doi.org/10.1007/978-1-4757-4145-2

[67] Bita Rouhani, Daniel Lo, Ritchie Zhao, Ming Liu, Jeremy Fowers, Kalin Ovtcharov, Anna Vinogradsky, Sarah Massengill, Lita Yang, Ray Bittner, Alessandro Forin, Haishan Zhu, Taesik Na, Prerak Patel, Shuai Che, Lok Chand Koppaka, Xia Song, Subhojit Som, Kaustav Das, Saurabh Tiwary, Steve Reinhardt, Sitaram Lanka, Eric Chung, and Doug Burger. 2020. Pushing the Limits of Narrow Precision Inferencing at Cloud Scale with Microsoft Floating Point. In *NeurIPS 2020*. ACM. https://www.microsoft.com/en-us/research/publication/pushing-the-limits-of-narrow-precision-inferencing-at-cloud-scale-with-microsoft-floating-point/

[68] Feras A. Saad, Cameron E. Freer, Nathanael L. Ackerman, and Vikash K. Mansinghka. 2019. A Family of Exact Goodness-of-Fit Tests for High-Dimensional Discrete Distributions. In *The 22nd International Conference on Artificial Intelligence and Statistics*. 1640–1649. http://proceedings.mlr.press/v89/saad19a.html

[69] Charbel Sakr, Yongjune Kim, and Naresh Shanbhag. 2017. Analytical Guarantees on Numerical Precision of Deep Neural Networks. In *Proceedings of the 34th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 70)*. PMLR, International Convention Centre, Sydney, Australia, 3007–3016. http://proceedings.mlr.press/v70/sakr17a.html

[70] Daniel Scharstein and Richard Szeliski. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision* 47, 1-3 (2002), 7–42. https://doi.org/10.1023/A:1014573219977

[71] Friederike Schmid and Nigel B. Wilding. 1996. Errors in Monte Carlo simulations using shift register random number generators. *International Journal of Modern Physics C* 6, 06 (1996), 781–787. https://doi.org/10.1142/S0129183195000642

[72] Claude E. Shannon. 1948. A mathematical theory of communication. *The Bell system technical journal* 27, 3 (1948), 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

[73] Priyesh Shukla, Ahish Shylendra, Theja Tulabandhula, and Amit R. Trivedi. 2020. MC2RAM: Markov Chain Monte Carlo Sampling in SRAM for Fast Bayesian Inference. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. 1–5. https://doi.org/10.1109/ISCAS45731.2020.9180701

[74] David B. Thomas and Wayne Luk. 2009. Using FPGA resources for direct generation of multivariate gaussian random numbers. In *Field-Programmable Technology, 2009. FPT 2009. International Conference on*. IEEE, 344–347. https://doi.org/10.1109/FPT.2009.5377680

[75] Madeleine B. Thompson. 2010. A comparison of methods for computing autocorrelation time. *arXiv preprint* (2010). arXiv:1011.0175

[76] Tijmen Tieleman. 2008. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*. 1064–1071. https://doi.org/10.1145/1390156.1390290

[77] Seth D. Tribble. 2007. *Markov chain Monte Carlo algorithms using completely uniformly distributed driving sequences*. Ph.D. Dissertation. Stanford University.

[78] Kush R. Varshney and Homa Alemzadeh. 2017. On the safety of machine learning: Cyber-physical systems, decision sciences, and data products. *Big data* 5, 3 (2017), 246–255. https://doi.org/10.1089/big.2016.0051

[79] Dootika Vats, James M. Flegal, and Galin L. Jones. 2015. Multivariate Output Analysis for Markov chain Monte Carlo. *arXiv preprint* (2015). arXiv:1512.07713

[80] Dootika Vats and Christina Knudson. 2018. Revisiting the Gelman-Rubin Diagnostic. *arXiv preprint* (2018). arXiv:1812.09384

[81] Aki Vehtari, Daniel Simpson, Andrew Gelman, Yuling Yao, and Jonah Gabry. 2015. Pareto smoothed importance sampling. *arXiv preprint* (2015). arXiv:1507.02646

[82] Shibo Wang and Pankaj Kanwar. 2019. BFloat16: the secret to high performance on cloud TPUs. *Google Cloud Blog* (2019). https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus

[83] Siyang Wang, Xiangyu Zhang, Yuxuan Li, Ramin Bashizade, Song Yang, Chris Dwyer, and Alvin R. Lebeck. 2016. Accelerating Markov Random Field Inference Using Molecular Optical Gibbs Sampling Units. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Seoul, Republic of Korea) *(ISCA '16)*. IEEE Press, Piscataway, NJ, USA, 558–569. https://doi.org/10.1109/ISCA.2016.55

[84] Yu Emma Wang, Yuhao Zhu, Glenn G. Ko, Brandon Reagen, Gu-Yeon Wei, and David Brooks. 2019. Demystifying Bayesian Inference Workloads. In *2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 177–189. https://doi.org/10.1109/ISPASS.2019.00031

[85] Max Welling and Yee Whye Teh. 2011. Bayesian Learning via Stochastic Gradient Langevin Dynamics. In *Proceedings of the 28th International Conference on International Conference on Machine Learning* (Bellevue, Washington, USA) *(ICML'11)*. Omnipress, Madison, WI, USA, 681–688.

[86] Yuling Yao, Aki Vehtari, Daniel Simpson, and Andrew Gelman. 2018. Yes, but Did It Work?: Evaluating Variational Inference. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, Stockholmsmässan, Stockholm Sweden, 5581–5590. http://proceedings.mlr.press/v80/yao18a.html

[87] Xiangyu Zhang, Ramin Bashizade, Craig LaBoda, Chris Dwyer, and Alvin R. Lebeck. 2018. Architecting a stochastic computing unit with molecular optical devices. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*. IEEE, 301–314. https://doi.org/10.1109/ISCA.2018.00034

[88] Stephanie Zierke and Jason D. Bakos. 2010. FPGA acceleration of the phylogenetic likelihood function for Bayesian MCMC inference methods. *BMC bioinformatics* 11, 1 (2010), 1–12. https://doi.org/10.1186/1471-2105-11-184