Characterizing Variability in Heterogeneous Edge Systems: A Methodology & Case Study

Hazem A. Abdelhafez, Hassan Halawa, Amr Almoallim, Amirhossein Ahmadi, Karthik Pattabiraman and Matei Ripeanu
Electrical and Computer Engineering
University of British Columbia
Vancouver, Canada
{hazem,hhalawa,amrm3lm,amirhmi,karthikp,matei}@ece.ubc.ca

***Abstract*—This study offers a methodology to characterize *intra-* and *inter-*node variability and applies it on two heterogeneous edge platforms (the NVIDIA Jetson AGX and Nano) for performance and power consumption. Firstly, we explore *intra-node* variability: investigate to what degree deployment decisions can limit it, highlight that it is unavoidable, and offer a scale so that one can compare to what other studies report. Secondly, we characterize *inter-node* variability by answering two questions: *(i)* Are the platforms we study statistically different in terms of the applications' power draw and runtime? and *(ii)* What is the magnitude of these differences? Finally, we attempt to answer the question of why is it paramount to characterize variability and take it into account? to achieve this, we discuss examples from the compiler and runtime optimization domains.**

I. Introduction

Context. Recent heterogeneous edge platforms (e.g., NVIDIA Jetson) employ a System on Chip (SoC) design that incorporates Processing Units (PUs) and memory modules that are configurable at runtime (e.g., by changing the operational frequency). As these devices are often used in resource constrained environments (e.g., battery powered), tuning the application and/or the platform configuration to maximize efficiency is essential [1], [2]. Such tuning relies on accurate characterization and/or modeling of the platform performance and power consumption to choose the best deployment configuration [1]–[4], compiler optimizations [5], [6], and/or to build performance/energy models. However, one major challenge often ignored in this context is: *variability*.

We define variability as a non-negligible difference between measurements of metrics of interest (e.g., power, runtime) under the same conditions (i.e., benchmark, software stack, hardware platform). Variability can manifest either over-time on the same node – i.e., *intra*-node variability, or across space, over multiple nodes – i.e., *inter*-node variability.

While variability has been exposed in other contexts (e.g., High-Performance Computing (HPC) [7]–[10] and Cloud Computing [11], [12] as we discuss in §II) in Edge / Internet of Things (IoT) contexts, variability has often been ignored even though it has the potential to be more significant. Reasons for an increased magnitude in these environments include: *(i)* devices that incorporate processing units with different architectures (each with a range of operational frequencies) which makes it more challenging to bin the devices based on their operating characteristics; *(ii)* a large number of deployed devices, even for a single application (e.g., traffic monitoring), which increases the chance of deploying supposedly 'identical' devices, yet sourced from multiple manufacturers and thus with some differences [13]; and *(iii)* deployment scenarios that are less controlled (e.g., no external temperature control).

Goals. We have two goals: firstly we aim to provide a methodology to characterize *intra-* and *inter-* node variability. Secondly we apply this methodology on two heterogeneous edge platforms (for performance and power consumption) and:

- Characterize *intra-node* variability and understand to what degree deployment decisions can limit it (§III). Additionally we preliminarily investigate the impact of ambient temperature variation (§VI-B).
- Characterize *inter-node* variability (§IV). To this end, our methodology focuses on two themes: *(i)* we explore whether the platforms we study are statistically different in terms of the applications' power draw and performance, and *(ii)* we explore various metrics to quantify the magnitude of these differences (§IV).
- Based on these characterizations, we argue that the observed variability matters form an application perspective (§VI).

Challenges. There are three main challenges in characterizing variability. *First, choosing sound statistical techniques (§IV).* With a few notable exceptions [5], [11], related work fails to use sound statistical techniques: for example, other studies use fixed-size samples without a rationale to support the chosen size, do not present confidence intervals/levels, offer only visual comparisons between distributions, and/or implicitly assume that the distributions of interest are Gaussian.

Second, carefully designing the experimental setup (§III). Over the past decade, prior work in HPC/cloud contexts (e.g. [7]–[11], [14]–[20]) has used a variety of ad-hoc experimental setups and data collection methodologies. As such, it is not possible to compare the results of these studies, or to understand if any of them offers a realistic lower bound for the variability that will be observed in practice. Our methodology proposes a rigorous experimental setup and data collection methodology, and offers a path to put past results in context.

Third, dealing with the large available configuration space (§V) particularly for modern heterogeneous edge platforms.

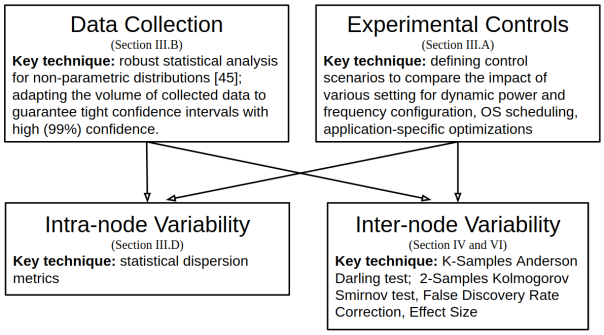

Fig. 1. Methodology roadmap and the key techniques employed by each of the building blocks of our methodology.

This stems from two factors: *(i)* a large number of hardware components that can be tuned (e.g., processing units, memory controller), and *(ii)* a wide range of available configurations for each component (e.g., an over 10x frequency range, with over a dozen levels for each component for the NVIDIA Jetson line). This makes it time consuming to cover the whole space when measuring the metrics of interest for multiple workloads.

Contributions. The main contribution of this paper is to present a methodology for studying the variability of edge platforms. We use this methodology to characterize intra- and inter-node variability on two popular edge platforms: the NVIDIA Jetson AGX and Nano. We use diverse workloads: ten Convolutional Neural Network (CNN) inference workloads, as well as kernels from the Rodinia [21] benchmark suite. The main contributions are:

- *A statistical methodology for characterizing variability.* We propose a generic, platform-neutral, statistically-sound methodology for studying intra- and inter- node variability (§III, §IV). Figure 1 presents an overview of its main components. Our methodology leverages well-known hypothesis testing methods: K-samples Anderson-Darling [22], 2-samples Kolmogorov-Smirnov [23], Benjamini–Hochberg False Discovery Rate correction [24]), and *effect* size estimation - *Robust d* [25]. These techniques are platform neutral, non-parametric (i.e., they do not make assumptions about the underlying distributions), and require only independent measurements sampled from the different characterized devices.
- *A characterization of intra-node variability.* (§III-D). We compare the impact of different controls that can be applied during application runtime to limit intra-node variability. Our experiments show a sizeable intra-node variability reduction (geomean: 5.2x for runtime and 5.8x for power) depending on the controls used. This characterization has multiple uses:
  - *A practical guide to application developers/deployers to reduce intra-node variability* based on the information that our characterization offers on the impact and the cost of various variability reduction measures.
  - *A unifying reference point to compare past work on variability* [7]–[11], [14]–[20] as different projects use different controls thus, their results are not directly comparable.
- *A characterization of inter-node variability.* (§IV). We show that even though the nodes are labeled by the manufacturer as identical (same SKU) they appear as statistically different in a majority of cases. We quantify the differences, i.e., we quantify the *effect size*, for multiple benchmarks across the entire frequency configuration range. For power, we find that differences are large (frequently higher than $\pm 10\%$ and up to $\pm 40\%$) and stable across benchmarking scenarios. For runtime, we find smaller and workload dependent differences (generally within the $\pm 6\%$ range, but sometimes up to $\pm 20\%$).
- *A preliminary characterization of the impact of temperature* variation on performance and power draw (§VI-B).
- *A discussion on the significance of the variability levels we find.* We present a discussion, based on real scenarios (§VII), that the reported variability in this study is impactful and necessitate accounting for it during application optimisation and deployment.

Workloads. The two categories of workloads we employ are commonly used to study the performance and power consumption of heterogeneous edge devices: *(i)* inference workloads using CNNs which are often deployed on edge platforms in contexts that have soft real-time constraints such as Computer-Vision (CV) applications. We note that CNN inference workloads are the main building blocks of several industry-backed benchmark suites (e.g., MLPerf [26], EdgeBench [27], AIMatrix [28], EdgeAIBench [29], and MLMark [30]). CNN workloads are complex aggregating more than a dozen different kernels with hundreds of individual kernel executions - thus they are less prone to the idiosyncrasies of the individual kernels or their launch contexts. *(ii)* Rodinia benchmarks which are often used to benchmark GPU-accelerated systems. In §V-B, we provide more details and expand on the rationale behind our choice of workloads.

II. Background

Classifying variability. Table I presents a classification of variability over two criteria. The first looks at where variability manifests: over time within the same node (*intra*-node variability), or over space across multiple nodes (*inter*-node variability). The second criterion looks at the source of the variability:

- *software*: e.g., interference with the OS and other applications [31], [32], program state initialization [5].
- *hardware*: e.g., process and manufacturing variations for components [13], [33].
- *environmental factors* such as thermal conditions [34], pressure, humidity, or supply voltage variations [35].

Implications of ignoring variability. The impact of Variability has been highlighted in multiple scenarios including: *(i)* user experience and satisfying Quality of Service (QoS) requirements in mobile applications [32], *(ii)* performance in large-scale Bulk-Synchronous Parallel (BSP) applications [38],

TABLE I
CATEGORIZING VARIABILITY.

	Software	Environment	Hardware
Intra-	■ interference with OS/applications [31], [32] ■ data and executable placement [5] ■ external systems (I/O, network)	■ temperature [34], ■ power source variations [35]	■ hardware aging [36]
Inter-	■ different driver / OS versions		■ manufacturing process variation [13], [33] ■ multiple sourcing [37]

or *(iii)* when building generic models for predicting the performance and power consumption of a platform [33]. In one example from the mobile computing domain, Wu et al. [31] show a significant performance variability for Machine Learning (ML) inference across mobile devices, and draw attention to the risk of poor user experience if deployments are optimized for the *average* observed performance. Wu et al. focus on intra-node variability only, and do not present a systematic methodology to characterize it.

Addressing variability. A first step towards addressing variability is to quantify it. Most characterization studies [39]–[46] are in the HPC domain. Among these, Weisbach et al. [47] propose a set of benchmarks based on a lightweight kernel to characterize performance variability in HPC systems. Kocoloski and Lange [8] propose Varbench as a set of benchmarks to precisely measure performance variability's impact on BSP applications. A large share of this related work, however, fails to use sound statistical techniques: for example, some studies use fixed-size samples without a rationale to support the chosen size, others do not present confidence intervals/levels, many offer only visual comparisons between distributions, and/or make implicit assumptions that the distributions of interest are Gaussian.

A second step is to incorporate variability in the design and development process. One way to achieve this is to build probabilistic models (e.g., for runtime estimation) that take variability into account to achieve better QoS guarantees [32]. Another approach is to dynamically adapt the runtime environment (e.g., scale processors' frequencies), or re-balance the workload [8] to handle variability. A more conservative approach is to optimize the system based on the worst case [31].

III. INTRA-NODE VARIABILITY

We start by focusing on intra-node variability with three complementary goals in mind: first, we aim to quantify the level of intra-node variability likely to be observed in real deployments. As multiple configuration / deployment decisions - e.g., enabling or disabling the default Dynamic Voltage and Frequency Scaling (DVFS) governor - impact variability, we aim to study their impact. Second, as prior work on variability lacks unified experimental controls, our study over a range of deployment decisions offers an avenue to put these past studies in a common context. Third, as we aim to study the impact of temperature variation (§VI-B), and inter-node variability, we need to find an experimental setup that minimizes the intra-node variability to mitigate its impact on the other two factors.

We adopt a methodology that consists of two key elements:

- *Controlling intra-node variability (§III-A).* We consider the various sources of intra-node variability incrementally to understand their impact. In particular we consider: *(i)* controls related to the platform's power profile configuration, *(ii)* controls related to DVFS settings adaptation; *(iii)* controls related to workload scheduling and temperature, *(iv)* PyTorch-specific controls, and *(v)* application-specific measures.
- *Measurement and data collection (§III-B).* We use a non-parametric sampling approach to collect representative data.

A. Methodology: Controlling Variability.

We define six *Control Groups* (CGs) to limit variability. Each group contains one or more controls that impact power consumption and/or runtime. These *control groups* are:

- **CG0** - *baseline controls that are included in all experiments*: *(i)* prevent any remote access or IO operations during data collection, *(ii)* place boards in the same ventilated rack in an air conditioned room (i.e., same ambient temperature), *(iii)* install the same software stack (i.e., benchmarks, libraries, drivers, OS image, etc.) on all the identical boards, *(iv)* remove unnecessary services that are pre-installed on the boards, and *(v)* limit concurrent user applications to one (the target benchmark).
- **CG1** - *controls related to the platform's power profile configuration, i.e., the power envelope).*: *(i)* modify the NVIDIA default power profile model (MAX-N) to allow all frequencies of the CPU and set it as the default power profile on all boards, *(ii)* prevent the CPU cores from going into idle state, *(iii)* use the default DVFS governor (e.g., *SchedUtil* for CPU and *nvhost_podgov* for GPU), and *(iii)* set the number of application threads to the number of cores available onboard (8 cores for the AGX).
- **CG2** - *controls related to dynamic frequency and voltage settings adaptation*: *(i)* set all components (CPU, GPU, and memory controller) governors to *userspace*, *(ii)* fix the components' frequencies to maximum, and *(iii)* disable rail gating for the CPU and GPU.
- **CG3** - *controls related to workload scheduling and temperature*: *(i)* isolate six out of the eight CPU cores for the target benchmark (i.e., core specialization), *(ii)* set the number of benchmark threads to six (i.e., the number of isolated cores), *(iii)* pin the benchmark threads to the isolated cores to prevent the OS from thread interruption and migration, and *(iv)* turn the fan on and set it to the maximum speed.
- **CG4** - *PyTorch-specific measures that impact the runtime performance*: *(i)* turn on the Just-In-Time (JIT) compiler optimization, and *(ii)* turn off gradient computations explicitly. Both can be done in PyTorch using a Python context manager.
- **CG5** - *additional application and PyTorch-specific measures with high performance impact (and thus unlikely to be used in practice)*: *(i)* set the underlying CUDA Deep Neural Network (cu-DNN) library algorithm selection to deterministic mode (i.e., always use the same algorithm for

the target kernel - e.g., convolution), and *(ii)* disable the cuDNN profiling phase for algorithm selection, and *(iii)* turn off the Python Garbage Collector (GC).

Based on these control groups we define five *control scenarios* (CS) where each *incrementally* incorporates one additional control group starting at CG_0. The general rule is that CS_i contains CG_0 to CG_i $\forall i \geq 1$ (*CG0* is the baseline for all control scenarios). Section III-D highlights the impact of the various *control scenarios* on intra-node variability.

B. Methodology: Data Collection

On each board, for each benchmark, and for each frequency configuration, we execute the following five-step sequence to collect a *sample* [1].

- **P1:** Reboot the board to always start from the same state.
- **P2:** Restart the characterization script (that runs the target benchmark) at least five times to eliminate any board-level cold-start effects (i.e., after the reboot in **P1**). We discard this initial data.
- **P3:** Run five warm-up iterations of the target benchmark to eliminate any application-level cold-start effects.
- **P4:** We call a *block* a set of repetitions of the segment of code that is benchmarked. We determine the number of repetitions (i.e., the block size) such that it meets three criteria: *(i)* the block contains at least 100 iterations of the target benchmark, *(ii)* the block runtime is at least one second, and *(iii)* the overhead of the timer and CUDA synchronization calls is $\leq 0.1\%$ of the block runtime.

  For runtime an *observation*[1] is the runtime of the block divided by the block size, while for power an *observation* is a power reading captured from the power sensors during the block runtime.
- **P5:** Collect multiple observations to fill a *sample*[1]. We collect as many observations as needed to reach a tight confidence interval for estimating the median. This criteria guarantees that the collected samples: *(i)* have low noise levels, as such, they are representative of their underlying, unknown populations, and *(ii)* allow us to uncover hidden patterns of inter-node variations as we show later in §IV. To achieve this, our data collection mechanism stops when the relative margin of error (RME)[2] for estimating the median is $\leq 0.5\%$ at the 99% confidence level (i.e., RME is the radius of the confidence interval). We collect at least 50 observations per sample (even when the RME threshold is satisfied with fewer).

The key features of our methodology are:

- *Platform independent.* We stress that the decision to stop the data collection for a sample (in **P5** below) does not make any assumption about the underlying distribution and, thus, it would not require any change to characterize a platform with different characteristics. This solution avoids the disadvantages of popular data collection approaches which often implicitly assume a Gaussian distribution.
- *Conservative.* To reduce possible noise introduced by inaccurate timers, each observation is the average of multiple runs of the target benchmark (see **P4** below). Thus our methodology offers an *extremely conservative* view of runtime variability. Our results should be seen as lower bound for what may be observed in practice.
- *Tight confidence intervals (with high confidence).* Typically in the literature a *fixed* number of observations is collected. This contrasts with our approach, which makes sure that a sample has enough observations to guarantee a narrow confidence interval for the estimate of the median (we aim for a range of 1% of the value of the median, that is 0.5% RME), with high confidence (99%). Thus, each sample accurately represents the underlying performance or power consumption of the target benchmark as noise has limited impact on the quality of the collected data.

C. Experimental Setup

We defer to a separate section (§V) the mundane parts of our experimental setup: the hardware platforms, the benchmarks, and additional details for data collection and experimental controls used in both intra- and inter-node variability experiments.

A note on temperature control and thermal throttling. We have placed all boards in a well ventilated enclosure. We monitor the on-board temperature sensors and make sure temperature is within a narrow 35 - 45C band for all experiments. According to the latest NVIDIA Jetson AGX thermal design guide [49], the maximum operating temperature limits (to operate without performance reduction) for the CPU, GPU, and other components are 86, 88, and 82°C respectively. Above these temperatures software or hardware throttling will reduce runtime frequencies to avoid overheating, thus reducing the board's performance, and impacting the reliability of the results. We also operate the on-board fans at the maximum speed all the time (they typically start operating automatically at ≈50°C only).

D. Results and Analysis

Metrics. We use two metrics to quantify variability:

- *QCV: the Quartile-based Coefficient of Variation* $QCV = 100 * (Q3 - Q1)/Q2$.
- *MADM ratio: the Median Absolute Deviation from the Median* ratio. For a sample X containing observations $x_0, x_1, ...x_n$, $MADM = median[x_i - median(X)]/median(X)$.

Both metrics are statistically robust [50]–[53] - with good performance for data drawn from a wide range of probability distributions, including non-parametric distributions. Dividing by the sample median allows us to calculate relative, dimensionless, variability scores so that we can compare across different control scenarios, and summarize results across multiple nodes.

[1] **Terminology used:** A *sample* is a set of *observations* (i.e., measurements for power or runtime at one frequency configuration). A *frequency configuration* is a unique combination of CPU, GPU and memory controller frequencies.

[2] **For a sample x of n observations**: $RME = 100 * |max(x[l], x[r]) - Q2|/Q2$ where Q2 is the sample median, x is sorted ascendingly, l and r are calculated based on the formula described in [48] as follows: $l = \lfloor \frac{n}{2} - z_{score} * \frac{\sqrt{n}}{2} \rfloor$, and $r = \lceil 1 + \frac{n}{2} + z_{score} * \frac{\sqrt{n}}{2} \rceil$

Results. Figure 2 shows the boxplot [3] of the *QCV* and *MADM* metrics across 14 AGX boards for the control scenarios defined in §III-A for one CNN: SqueezeNet. Table II shows the median value for *QCV* across the 14 AGX boards for each network (the trends for MADM or other cutoff points in the distribution are similar thus, due to limited space, we do not present them).

Takeaways. The results highlight a number of observations:

- *Intra-node variability manifests itself for both runtime and power* in spite of our conservative measurement approach, and tight controls used to limit it. We stress that these results are lower bound estimates for the intra-node variability that will be observed in practice given that not all the controls - even the most basic ones included in CG0 and thus included in all our experiments - can always be used for reasons related to their performance impact or deployment scenario constraints (e.g., completely disabling advanced power saving features, co-deployed applications, or environments that limit the temperature variations).
- *Deployment and configuration decisions have a sizeable impact* and can reduce the observed intra-node variability for both runtime and power. Comparing *CS1* and *CS5* (*CS4* for power) the geomean variability reduction is 5.2x for runtime (5.8x for power). There is some variation depending on the specific application – e.g., variability reduction can be as high as 10x for some applications (e.g., VGG, ShuffleNet).
- *CS4 is a potential middle ground between a viable level of control and performance/power impact*. To substantiate this claim we present in detail an analysis for SqueezeNet.
  - *Performance.* Starting with CS1 performance tends to improve gradually and reaches its peak at CS4. The reasons are that in CS4: *(i)* we fix the dynamic frequency and power settings of the boards, thus guaranteeing maximum possible performance during all the application phases (i.e., no frequency scale down at any stage), *(ii)* we isolate CPU cores and pin threads to these cores, thus eliminating any thread scheduling or migration by the OS and improving locality, and *(iii)* we enable PyTorch-specific performance improvements. These measures improve the performance, and also limit variability as the system has limited dynamic control on the runtime conditions. As such, both variability scores (QCV and MADM) decrease gradually starting with CS1 till CS4. In CS5, we disable the dynamic cuDNN optimizer that picks the best algorithm for some kernels (e.g., convolution) during runtime. Instead, it uses a fixed algorithm. This leads to a significant drop in performance (70% increase in runtime for CS5 compared to CS4) yet also to a significant variability reduction (by 68% for QCV from CS4 to CS5).
  - *Power consumption.* From CS1 to CS3, power consumption varies marginally but CS1 exhibits the highest variability score due to the dynamic nature of DVFS default scheduler. CS4 draws, by far, the most power amongst all control scenarios as it disables rail gating, fixes frequencies to maximum, and enables resource-demanding optimizations. CS5 consumes the least amount of power (40% decrease from CS4). We believe this happens due to a lower resource utilization by the non-optimum fixed algorithm selection for the underlying kernels in CS5.
- *Disabling cuDNN optimizations for kernel selection has multiple side effects on individual benchmarks*: lower variability for most of them, higher variability for a few, and a reduction in absolute performance for all networks.

[3]**Box plots interpretation:** For all the box plots presented in this paper the top and bottom sides represent the first and third quartiles (Q1 and Q3). The horizontal line represents the median (Q2). The bottom/upper fences represent the 10^{th} and 90^{th} percentiles of the data respectively. The outliers are removed for legibility.

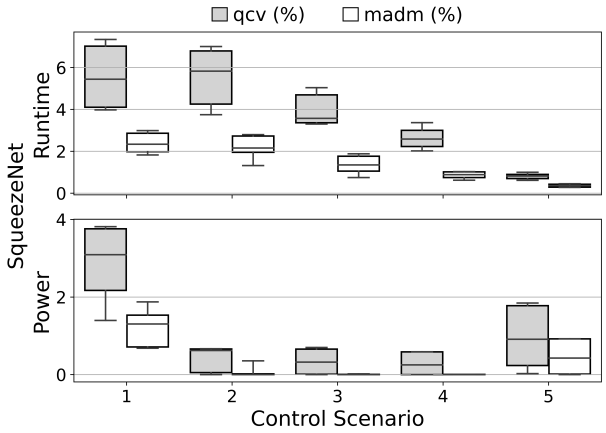

Fig. 2. Control scenarios' impact on variability (for SqueezeNet). The X-axis is the *control scenario* ID. The Y-axis presents the *QCV*/*MADM*. Boxplots[3] summarize data collected from 14 Jetson AGX boards.

TABLE II
MEDIAN *QCV* VALUES FOR DIFFERENT CONTROL SCENARIOS FOR EACH NETWORK. THE MAX/MIN COLUMN INDICATES THE RATIO BETWEEN THE MAXIMUM AND MINIMUM QCV VALUES ACROSS ALL CONTROL SCENARIOS FOR EACH NETWORK (THE ∞ VALUES ARE EXCLUDED IN THE MEAN AND GEOMEAN CALCULATIONS).

Benchmark	Runtime						Power					
	CS1	CS2	CS3	CS4	CS5	Max/ Min	CS1	CS2	CS3	CS4	CS5	Max/ Min
AlexNet	0.52	1.55	1.47	1.17	0.55	2.97	1.27	0.40	0.52	0.55	0.74	3.14
DenseNet	0.45	0.09	0.07	0.05	0.55	11.12	1.27	0.90	0.61	0.59	1.26	2.16
GoogeLeNet	5.07	2.91	2.58	2.52	1.32	3.85	3.34	0.43	0.89	0.72	1.23	7.78
Inception3	0.67	1.31	0.68	0.35	1.47	4.18	0.97	0.63	0.61	0.61	1.12	1.86
MnasNet	5.76	3.81	3.33	2.82	0.63	9.10	1.43	0.93	0.00	0.03	1.17	∞
MobileNetV2	4.12	3.81	3.00	3.25	2.80	1.47	1.44	0.03	0.002	0.75	0.02	589.20
ResNet	0.91	2.24	1.89	1.99	1.96	2.46	1.33	0.64	0.62	0.60	0.92	2.21
ShuffleNetV2	12.67	3.84	3.87	2.96	1.18	10.77	0.01	0.02	0.01	0.02	0.01	2.02
SqueezeNet	5.44	5.83	3.57	2.59	0.82	7.09	3.09	0.62	0.32	0.25	0.91	12.40
VGG	0.79	0.15	0.18	0.13	0.07	10.74	1.16	0.46	0.46	0.48	0.52	2.50
GeoMean	1.98	1.51	1.23	1.00	0.82	5.20	0.97	0.32	0.00	0.29	0.42	5.84
Mean	3.64	2.55	2.06	1.78	1.14	6.38	1.53	0.51	0.40	0.46	0.79	69.25

IV. INTER-NODE VARIABILITY

We shift our focus to inter-node variability characterization. We show that for most frequency configuration points the nodes appear as statistically different (for both performance and power) (§IV-A) and we offer a metric to quantify this difference (§IV-B).

Experimental setup highlights. We stick with our unconventional presentation structure: we present below the experimental setup elements that are essential to understand and appreciate the inter-node variability characterization, while the more mundane parts of the experimental setup are presented in §V.

The Jetson boards are configured with *CS4* control scenario used for intra-node variability characterization (§III-A) for CNNs workloads (*CS3* is used for Rodinia as *CS4* and *CS5* are not applicable). The motivation behind this choice is that this control scenario limits intra-node variability as much as possible without sacrificing application-specific performance optimizations (e.g., cuDNN algorithm selection) or employing impractical controls (e.g., disabling Python's GC) - thus it is the likeliest to be adopted in practice if one focuses on limiting variability.

We use the data collection methodology described in the previous section (§III-B). For each *benchmark* we collect one sample[1] for power and one for runtime, at each frequency configuration point on each board. This leads to about 205,000 samples and at least 10 million observations on the Jetson AGX, and about 160,000 samples, and at least 8 million observations on the Nano.

A. Q1: Are the boards statistically different?

Methodology. To answer this question, we leverage *statistical significance tests*. These tests are generally used in two contexts: *(i)* to determine if a sample from a certain population belongs to a specific parametric distribution [54] (i.e., goodness-of-fit test), or *(ii)* to determine whether samples drawn from different populations belong to the same distribution. We are interested in the latter: for a chosen configuration point, we extract a sample (power consumption or runtime) on each board and use the statistical significance tests to understand whether the boards appear as identical (i.e., do the samples likely come from the same distribution?).

For each test, there is a *null hypothesis* (e.g., that the provided samples belong to the same distribution). We test whether we can reject the null hypothesis with high confidence (99%). We use a multi-step analysis consisting of three tests:

- First, the K-samples Anderson-Darling test [22], [54], allows testing multiple nodes at a time (i.e., the null hypothesis in this case is that *all* K samples drawn from the K nodes belong to the same distribution).
- Second, since the above test would reject the null hypothesis even with one defective node, we use the two-samples Kolmogorov-Smirnov (KS) test [23] and compare nodes pairwise (i.e., the null hypothesis in this case is that the samples drawn from nodes A and B belong to the same distribution).
- Third, as we run the KS test to test multiple null hypotheses (i.e., for multiple frequency configurations and benchmarks), we apply the Benjamini–Hochberg False Discovery Rate (BH-FDR) multiple-test correction to the two-samples KS test p-values to control the rate of false-positives (i.e., incorrectly rejecting the null hypothesis).

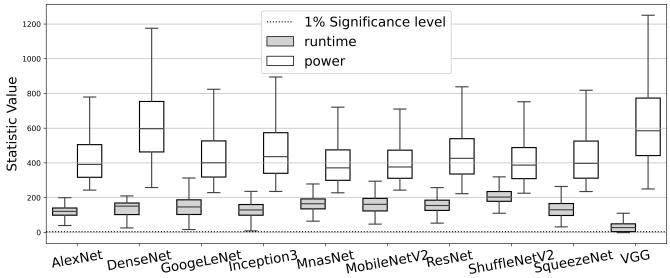

Fig. 3. Boxplots[3] present K-samples Anderson-Darling test score for the inference task of ten different CNNs on the Jetson AGX for over 450 different frequency configurations. The outliers are omitted for legibility. The dashed horizontal line indicates the critical value for 99% confidence level. (Running on Nano or using Rodinia benchmarks show similar trends but are omitted due to space limitations).

Results: K-samples Anderson-Darling Test. For each frequency configuration and benchmark, we run the K-samples Anderson-Darling test (Fig. 3 presents this for AGX). We find that, with 99% confidence, the null hypothesis can be rejected for virtually all the configurations sampled for both power and runtime for both AGX and Nano.

Results: Two-samples Kolmogorov-Smirnov Test. The K-samples AD test indicates that at least one board is different. This could be a defective board, so we turn to the two-samples Kolmogorov-Smirnov test to compare the boards pairwise. We run the test for each configuration and *benchmark*, and for each board pair: this leads to about 42K tests for AGX (19K for Nano) for each benchmark. To limit the false positives when testing multiple hypotheses we apply the Benjamini–Hochberg [24] False Discovery Rate correction.

Tables III and IV show the ratio of the tests where the *null hypothesis* (i.e., the samples belong to the *same* distribution) can be rejected. On average, across all benchmarks, the results indicate that the boards are statistically different: *the null hypothesis can be rejected with 99% confidence for most pairwise comparisons*: on the AGX over 99% for power consumption and 82% for runtime; while on the Nano the corresponding ratios are 77% and 79%.

B. Q2: How significant is the difference between boards?

Methodology. Using the statistical significance tests we demonstrated that for a wide majority of frequency configurations and benchmarks, boards are indeed statistically different (a 'yes/no' question). However, statistical significance tests do not indicate the *magnitude* of the difference; i.e., they do not indicate the *effect size* [55].

One of the most popular techniques to estimate the effect size is the *Cohen's d* [56]. It measures the effect size as the difference between two samples' means divided by their pooled standard deviations. This technique, however, assumes that both samples are drawn from normal distributions and have the same variance. Since these assumptions do not hold in our case, we use the *scaled robust d* (d_r) metric [25], a modification of *Cohen's d* that is robust [55]. Similar to *Cohen's d*, the *scaled robust d* (d_r) compares the differences

TABLE III
PERCENTAGES OF TESTS IN WHICH THE P-VALUES INDICATE THAT THE NULL HYPOTHESIS CAN BE REJECTED ON THE JETSON AGX (I.E., SAMPLES DO *not* BELONG TO THE SAME DISTRIBUTION) FOR TWO DIFFERENT CONFIDENCE LEVELS (99% AND 95%).

	Conf	AlexNet	DenseNet	GoogleNet	Inception3	MnasNet	MobileNetv2	ResNet	ShuffleNetv2	SqueezeNet	VGG	Mean
Power	1%	99.84	99.84	99.76	99.81	99.80	99.78	99.82	99.59	99.90	99.89	99.80
	5%	99.86	99.86	99.79	99.85	99.83	99.81	99.86	99.64	99.92	99.93	99.84
Runtime	1%	84.24	85.28	86.21	83.54	92.88	90.65	90.95	94.95	84.95	28.46	82.21
	5%	88.70	89.15	90.24	88.15	95.29	93.52	93.96	96.69	89.28	34.01	85.90

TABLE IV
PERCENTAGES OF TESTS IN WHICH THE P-VALUES INDICATE THAT THE NULL HYPOTHESIS CAN BE REJECTED ON THE JETSON NANO (I.E., SAMPLES DO *not* BELONG TO THE SAME DISTRIBUTION) FOR TWO DIFFERENT CONFIDENCE LEVELS (99% AND 95%).

	Conf	AlexNet	DenseNet	GoogleNet	Inception3	MnasNet	MobileNetv2	ResNet	ShuffleNetv2	SqueezeNet	VGG	Mean
Power	99%	98.18	63.75	98.94	89.79	98.51	97.88	98.26	98.9	98.02	48.65	89.09
	95%	99.16	75.36	99.42	93.85	99.18	98.74	99.19	99.46	98.91	60.73	92.4
Runtime	99%	98.13	93.75	98.43	96.51	97.25	96.66	98.38	62.27	74.39	87.73	90.35
	95%	98.76	95.09	99.1	97.3	98.01	97.38	98.94	70.94	80.94	91.29	92.77

between the means of two samples relative to their pooled standard deviation. More intuitively, a large d_r value indicates that the means of the two distributions are far apart relative to their observed variance.

Below, we use the *scaled robust d (d_r)* estimate of the distance between distributions in two ways: first, in a relative way, we use it to characterize differences between all the node pairs (at various configuration points) and show there is a large range of differences between nodes. Second, we use it in an absolute way, to study how many node pairs are significantly dissimilar - by choosing a particular threshold for d_r.

Results. Fig. 4 shows the distribution of the d_r values on the AGX. The figure highlights that there is significant spread in the d_r values (particularly for power).

While the interpretation of this metric is context specific, a first intuition is provided by the guideline originally provided by Cohen for interpreting d values: ± 0.25, ± 0.5, and ± 1.0 values represent thresholds for small, medium, and large effect sizes, respectively. With this interpretation, on average across all benchmarks, on the AGX platform, 77.1% of board pairs and configurations we observe *large* differences for power and 33.1% for runtime.

Even with a more conservative threshold of ± 2.0, *large* differences are observed, on average, about 72.8% for power and 24.38% for runtime.

Takeaways. This analysis indicates that power consumption and performance behavior appear as statistically different across most pairs of 'identical' boards (on both AGX and Nano). The d_r metric suggests that the magnitude of the difference is uniformly large for power and smaller (and less evenly distributed) for runtime. Section §VI offers a deeper exploration of the observed inter-node variability and highlights its possible consequences.

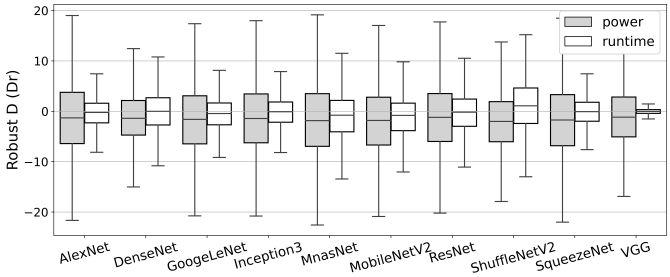

Fig. 4. Boxplots[3] presenting pairwise d_r values distribution across ten networks on the Jetson AGX (Rodinia benchmarks show similar trends but are omitted due to space limitations). The guideline for interpreting the effect size is: ± 0.25, ± 0.5, and ± 1.0 represent thresholds for small, medium, and high effect sizes respectively.

V. EXPERIMENTAL SETUP

A. The Experimental Platform

The hardware platform We focus on two popular NVIDIA heterogeneous edge platforms: the Jetson AGX, and the Jetson Nano. Both combine NVIDIA's state-of-the-art GPU technology with low-powered ARM CPU cores in a shared-memory architecture to deliver massive compute power and high energy efficiency in a tiny physical footprint.

Jetson AGX. The most important feature of the AGX in our context is that it provides a wide (10x) range of frequencies for the main processing elements and the memory controller. Each CPU core's operating frequency can range from 115.2MHz to 2.265GHz at a fine granularity (with 29 supported CPU frequency levels). The Volta-based GPU has 512 CUDA cores, and 64 Tensor cores. The CUDA cores support dynamic frequency scaling between 114.75 MHz and 1.377GHz at a fine granularity (with 14 frequency levels). Finally, the AGX allows dynamic frequency scaling for the External Memory Controller (EMC) - between 204MHz and 2133MHz (with 9 frequency levels). The frequency state space of these three components has a total of $\approx 3.6K$ combinations.

Jetson Nano: Similarly, the Nano supports a range of frequencies for its CPU, GPU, and memory controller. Each of the four CPU cores has an operating frequency ranging from 102.0MHz to 1.479GHz (15 CPU frequency levels). The Maxwell-based GPU has 128 CUDA cores, supports dynamic frequency scaling between 76.8MHz to 921MHz (with 12 frequency levels). The Nano's EMC supports dynamic frequency scaling for two levels, 204MHz and 1600MHz. The

frequency state space of these three components has a total of 360 combinations.

Table V provides an overview of the Jetson AGX and Nano features.

Dealing with the large configuration space On the AGX board, there are $\approx 3.6K$ unique combinations (i.e., configurations) of CPU, GPU, and memory controller frequencies. We use sampling to reduce the space by almost one order of magnitude: we sort each component's (e.g., CPU, GPU) frequency range, and sample every other frequency. This sampling scheme reduces the space to 525 configurations, yet covers most of the frequency range of each component. On the Nano, we sample the largest twelve (out of 15) CPU frequencies, 9 (out of 12) GPU frequencies, and all memory controller frequencies (2). This reduces the space from 360 configurations to 216 configurations.

The software stack *The NVIDIA JetPack.* We use the NVIDIA JetPack SDK version 4.4 across all AGX boards (v4.5 on the Nano). It includes the latest Linux OS for Tegra and driver package (L4T v32.4.6) (v32.5.1 on the Nano) for the Jetson platform. It also incorporates a full set of optimized software libraries (e.g., CUDA v10.2) to build applications targeting the various on-board PUs (e.g., GPU, deep learning accelerators, computer vision).

B. The Workload

We focus on two different workloads: *(i)* ML *inference* models based on CNNs, and *(ii)* a sub-set of Rodinia [21] benchmarks.

CNNs are one of the most widely used categories of ML models. The motivation behind choosing CNNs to benchmark the variability of edge platforms is twofold: *(i)* they power several computer vision related tasks such as object recognition and classification in several edge applications [31] [59] (e.g., drones, smart home, surveillance, etc.), and *(ii)* they are the main building blocks of several industry-backed benchmarks (e.g., MLPerf [26], MLMark [30], EdgeBench [27], AIMatrix [28], EdgeAIBench [29]).

Machine learning: PyTorch framework. We use PyTorch [60], version 1.6, as it is one of the most widely used deep learning frameworks. It allows developers and researchers to build deep learning networks, mainly using the Python programming language. The core functionality of PyTorch is implemented in C++ and exposed as Python C++ extensions to the users (i.e., LibTorch).

Machine learning: Pre-trained CNNs. We use Torchvision [61], version 0.8, to load the pretrained CNNs used in the experiments. Torchvision is a popular machine-vision package - compatible with and part of the PyTorch project - that encompasses popular data sets, models, and image transformations to process visual data. All classification CNNs included in Torchvision are trained and optimized on the 1000-class ImageNet dataset. Each vision network accepts inputs of a specific size (i.e., 4D Tensor). We generate a shape-compatible input tensor for each network (with randomly generated numbers), and use it during the inference task. We study ten different classification networks from Torchvision: AlexNet, GoogleNet, ResNet, MobilenetV2, MNASNet, SqueezeNet, Shuffle-NetV2, DenseNet161, VGG16, and InceptionV3.

Rodinia. Rodinia benchmarks are often employed in characterizing the performance of GPU-accelerated systems [21]. The benchmarks represent different categories of applications (e.g., stencil, dynamic programming) that stress different components of the underlying computing resources. Additionally, Rodinia benchmarks offer an alternative workload to PyTorch-based CNNs with a shallower software stack, thus allowing us to explore the correlation between software stack complexity and variability, if any. We use six applications from Rodinia benchmark [21]: Lower-upper Decomposition (LUD), Hotspot3D, Breadth-first-search (BFS), Nearest Neighbor (NN), Needleman-Wunsch (NW), and Huffman coding. We choose this subset because it represents different categories of computational patterns (e.g., structured grid, graph traversal, dynamic programming, etc.) from different domains.

C. Power and Time Measurements

Timing measurements. We use the CUDA Events API [62] for precise timing measurements (0.5µs resolution). We include CPU-GPU memory transfers in the timing and power measurement of GPU-based applications.

Power measurements. For the AGX platform, we use the two on-board INA3221 [63] ($\pm 5\%$ error) Power Monitoring Units (PMU) that can be read via an exposed virtual file system (sysfs). PMUs have six power 'rails' (for CPU, GPU, memory module, computer vision (CV) accelerator, auxiliary on-chip components, system IO) which measure the power each component draws. We discard the CV accelerator and system IO rails as they are not relevant to our experiments. The Nano has one on-board INA3221 PMU with three power rails' (for the CPU, the GPU, and the overall main module input respectively), of which the CPU and GPU rails are used. This PMU has a ($\pm 5\%$ error) for measurements above 0.2 Watts, and ($\pm 15\%$ error) for measurements below that. We sample the power sensors with a $2Hz$ sampling rate.

We have confirmed that the PMUs' (in)accuracy is not the main element driving the observed variability. For example, we confirmed the conclusions regarding inter-node variability using an external power meter (WattsUp Pro Kit [64]) on a sample consisting of five Jetson AGX boards. The external meter results are inline with the results collected from the on-board PMUs. The only difference is a slight reduction in the absolute differences that we attribute to the additional components (e.g., power adaptor losses, flash storage, networking and other peripherals circuitry) being measured by the external power meter.

VI. Discussion

We explore two additional topics. First we aim to better understand the magnitude of the inter-node variability (§VI-A). Our exploration leads to the following conclusion: for both power and runtime, differences between nodes are sizeable

TABLE V
NVIDIA JETSON AGX AND NANO SPECIFICATIONS AS REPORTED BY THE OS AND RELEVANT DOCUMENTATION [57], [58]

	AGX Platform	Nano Platform
CPU	8-cores ARM v8.0 64-bit CPU	4-cores ARMv8 64-bit CPU
Architecture	ARMv8-A	ARMv8
L1d/L1i/L2/L3 Cache	64KB/128KB/2MB/4MB	32KB/48KB/2MB/NA
Min/Max Core Frequency Range	**115.2MHz → 2.265GHz**	**102.0MHz → 1.430GHz**
Peak Theoretical FLOPS (SP)	144.96 GFLOPS	45.76 GFLOPS
GPU	Volta: 4 TPCs — 8 SMs — 512 CUDA Cores — 64 Tensor Cores	Maxwell: 4 TPCs — 8 SMs — 128 CUDA Cores — 64 Tensor Cores
L1/L2 Cache	128KB/512KB	128KB/512KB
Frequency Range	**114.75MHz → 1.377GHz**	**76.8MHz → 921MHz**
Peak Theoretical FLOPS	1.4 TFLOPS	512 TFLOPS
DRAM	16GB LPDDR4x (2133MHz, 2 Channels)	4GB LPDDR4x (1600MHz, 4 Channels)
Data Bus Width	256bit	64bit
Min/Max EMC Frequency Range	**204MHz → 2133MHz**	**204MHz → 1600MHz**
Peak Theoretical Bandwidth	136.512 GB/sec	25.6 GB/sec

and reproducible across runs, which *indicates an intrinsic difference between the boards*. For power, the differences are particularly large and directionally stable regardless of the benchmark and frequency configuration used. For runtime, differences are relatively smaller, however, they are still significant, and impactful if ignored, as we argue in this section, and in the next section - §VII).

Second, we aim to explore the impact of temperature variation on performance and power draw (§VI-B). Our preliminary data suggest a sizeable impact of temperature variation on both power and runtime - roughly of the same magnitude as the inter-node variability effects.

A. In-depth inter-node variability analysis

One drawback of effect-size metrics including the *scaled robust-d* we use in §IV-B is that interpreting the obtained values is domain specific. To shed more light, here we examine whether there are large differences between boards using an alternative metric: relative difference $RD\%$.

Data processing. For each board, for each frequency configuration and benchmark we compute the median m for each sample[1] collected (§III-B). Then, across all boards, for the samples collected for the same application and frequency configuration we compute their median of medians M. Finally, we go through all the observations of all samples and, to remove any statistical outliers on a per frequency configuration basis, we select only the observations that are within the $\pm 0.5\%$ from their sample's median (m). For the observations O selected, we compute two values: *(i)* the relative difference to the sample's median $rd = 100 * (O - m)/m$, and *(ii)* the relative difference to the median of medians $RD = 100 * (O - M)/M$. We then group the rd (i.e., intra-node variability) and RD (i.e., inter-node variability) values per benchmark and board, and use boxplots[3] (the whiskers' limits are extended to plot all values as we get rid of the statistical outliers on a per frequency basis earlier) and scatter plots to present rd and RD distributions in Fig. 5 (We present a subset of the benchmarks due to space limitations; other benchmarks and experiments on the Nano lead to qualitatively similar observations).

Power. We make two observations:

- *(i) large magnitude* - the RD analysis indicates that there are frequently large differences between boards. The relative differences are frequently larger than 10% (for some node pairs even the difference between medians is larger than 10% for all benchmarks); and differences can get as large as 70% (for nn and lud benchmarks - not shown).
- *(ii) directionally stable and workload independent* - a node will consistently draw more/less power compared to others (e.g., node-A always consumes less power than node-B regardless of benchmark and frequency configuration point), which indicates that the observed variability is likely due to inherent differences between the studied boards.

As the levels of power consumed are, depending on the frequency configuration, between a few watts to 10s of watts, these differences will matter in a wide range of real world scenarios where decisions based on power (or energy) are made. Such scenarios range from battery provisioning to controllers aiming to minimize power consumption for surveillance cameras by tuning the frequency configuration [1], [65].

Runtime. For runtimes, as hinted by the analysis in §IV, the inter-node differences are smaller. Since a trend is less striking visually in Fig. 5, we investigate the distribution of pairwise RD differences. Table VI presents the following analysis: out of the over 41k node-pairs and configuration points we have for each benchmark, the table presents for how many of these is the median (left two columns) or the maximum (right two columns) RD difference higher than a certain threshold (3% and 5%). We use the data cleaning process described earlier and used in Fig. 5. Based on Fig. 5 and Table VI we make the following observations:

- *(i) situations with large pairwise node differences exist* for specific benchmarks and frequency configurations yet they are not frequent. RD differences can reach up to 20% (e.g., nw and $ShuffleNetV2$), yet such large values appear for few frequency configurations.
- *(ii) smaller, yet significant, differences between nodes are relatively frequent*. Table VI shows that on average (geomean) across CNN benchmarks 4.78% of the data points have a RD difference between medians larger than 3% (and 0.44% larger than 5%). The differences are more frequent and more pronounced for Rodinia kernels where 7.53% of the data points have a RD difference between medians larger than 3% (and 3.20% larger than 5%). Some benchmarks (e.g., $GoogleNet$, $ShuffleNetV2$, $huffman$, nn and nv) expose larger inter-node differences more fre-

TABLE VI

PERCENTAGE OF NODE PAIRS WITH A PAIRWISE RELATIVE DIFFERENCE RD ABOVE A CERTAIN THRESHOLD FOR RUNTIME. THE *Medians* COLUMNS (LEFT TWO) COMPARE THE MEDIANS OF TWO CORRESPONDING SAMPLES FROM A NODE PAIR. THE *Maximum Difference* COLUMNS (RIGHT TWO) REPRESENT A WORST CASE SCENARIO BY USING THE FARTHEST AWAY OBSERVATIONS FROM THE COMPARED SAMPLES (AFTER DATA CLEANING). VGG IS REMOVED FROM THE GEOMEAN CALCULATIONS.

Benchmark		Medians (%)		Max Difference (%)	
		5%	3%	5%	3%
CNNs	AlexNet	0.03	1.25	0.24	3.38
	DenseNet	0.04	1.43	0.19	3.61
	GoogeLeNet	2.33	10.18	4.66	17.87
	Inception3	0.22	2.80	0.82	6.50
	MnasNet	1.58	8.58	3.38	15.79
	MobileNetV2	0.64	6.73	1.90	13.58
	ResNet	3.07	8.90	4.89	14.43
	ShuffleNetV2	8.14	23.25	12.49	36.43
	SqueezeNet	0.04	2.15	0.31	6.25
	VGG	0.00	0.00	0.00	0.23
	GeoMean	0.44	4.78	1.40	10.00
Rodinia	bfs	5.95	8.75	6.58	10.43
	hotspot3d	0.03	0.57	0.03	0.88
	huffman	17.95	27.24	21.54	32.84
	lud	1.21	3.62	1.85	5.77
	nn	12.94	14.50	13.56	16.90
	nw	21.38	25.60	22.55	26.74
	GeoMean	3.20	7.53	3.66	9.61
All	GeoMean	0.97	5.74	2.06	9.84

quently. (Table VI).

- *(iii) workload-dependent* - a node may appear faster for one benchmark and slower for another. We believe that this stems from the fact that different workloads stress the underlying components (e.g., GPU, Memory) differently.

For both power and runtime, we stress the following: *(i) reproducibility* - repeating the characterization experiments will lead to the same results and conclusions (we verified this on a subset of the frequency configurations and benchmarks), *(ii) representativeness* - the intra-node variability rd values (left boxplots in Fig. 5) are extremely low, which indicates the success of our proposed measurement approach to mitigate noise and suggests that the collected data adequately represents the true performance and power consumption behavior of the boards, and *(iii) significance* - the reported RD values extend beyond magnitudes that related work deemed detrimental for QoS guarantees or clean experimental design (we dedicate section VII to discuss this aspect in more detail).

B. The impact of ambient temperature

So far all experiments have controlled temperature in a narrow range. We have gathered preliminary data to quantify the impact of a large temperature variation. We apply the same methodology and experimental controls as in the rest of this paper with the exception that we use an older NVIDIA Jetson TX2 board and we use only the maximum frequency. We compare two settings: 25°C and 70°C ambient temperature by placing the board in a controlled temperature enclosure [66]. We verified that throttling does not kick in even at 70°C. We use a control scenario similar to *CS4*.

TABLE VII

TEMPERATURE IMPACT ON RUNTIME. THE FIRST TWO COLUMNS PRESENT THE MEDIAN RUNTIME AT 25C AND 70C. THE SECOND TWO COLUMNS PRESENT THE MEDIAN SLOWDOWN AT 70C, AND THE THRESHOLD FOR THE 3RD QUARTILE FOR SLOWDOWN

Model	at 25°C (ms)	at 70°C (ms)	Slowdown (median) (%)	Slowdown (3rd quartile) (%)
Squeezenet	12.97	13.15	1.40%	1.57%
Mobilenetv2	12.24	12.44	1.61%	2.42%
Alexnet	13.77	14.48	5.15%	5.44%
Googlenet	16.48	16.73	1.56%	2.54%
Resnet	14.69	15.06	2.53%	3.00%
Shufflenetv2	14.58	15.06	3.28%	3.81%
GeoMean			**2.30%**	**2.91%**

Impact on power draw. As expected the impact is sizeable: on average, over the benchmarks we have studied, 18.49% higher power consumption at 70°C than at 25°C. We can attribute roughly half the increase to higher static power draw, and the other half to higher dynamic power draw.

Impact on runtime. More surprising is the impact on runtime. Table VII presents the temperature effect on runtime. To eliminate the application restart effects, we run each experiment multiple times (i.e., collect the median of each experiment) until we estimate the median of medians within ±0.5% with 99% confidence, then compute the median slowdown (by comparing the median of the medians at 25°C and 70°C); and the 1st quartile of the slowdown distribution (by comparing Q1 at 25°C and Q3 at 70°C). Runtime at 70°C is, on average, 2.3% slower, with a maximum slowdown of 5.15% for AlexNet. Q3 slowdown is on average 2.93%, with a maximum slowdown of 5.44% for AlexNet.

VII. WHY DOES THIS MATTER?

While sections III and IV detail our methodology and characterize variability on two popular edge platforms, this section is driven by two interrelated questions: *Q1: Why is it paramount to characterize variability and take it into account?*, and *Q2: Are the levels of variability we find high enough to matter?*.

The issue is that in most cases [4] researchers and practitioners implicitly expect that an application will have the same runtime (or power draw) for different runs (on the same node or on identical nodes). As our characterization demonstrates this assumption does not hold: *(i)* the variability we find is sizeable (and likely a lower bound for what is observed in practice); *(ii)* some of the factors driving variability are unavoidable; and *(iii)* the impact of various factors that drive variability (e.g., inter-node differences, temperature) is cumulative.

Below we use a couple of examples to highlight the importance of characterizing variability and to highlight that the levels of variability we find are large enough to matter. The first example highlights the challenge of making decisions based on direct measurement (in the context of compile-time optimizations), while the second example highlights the

[4] A study by Mytcowicz et al. [5] confirms that this is common practice: they surveyed 133 papers presented at top-tier conferences and found that none of them properly account for (what we call) intra-node variability.

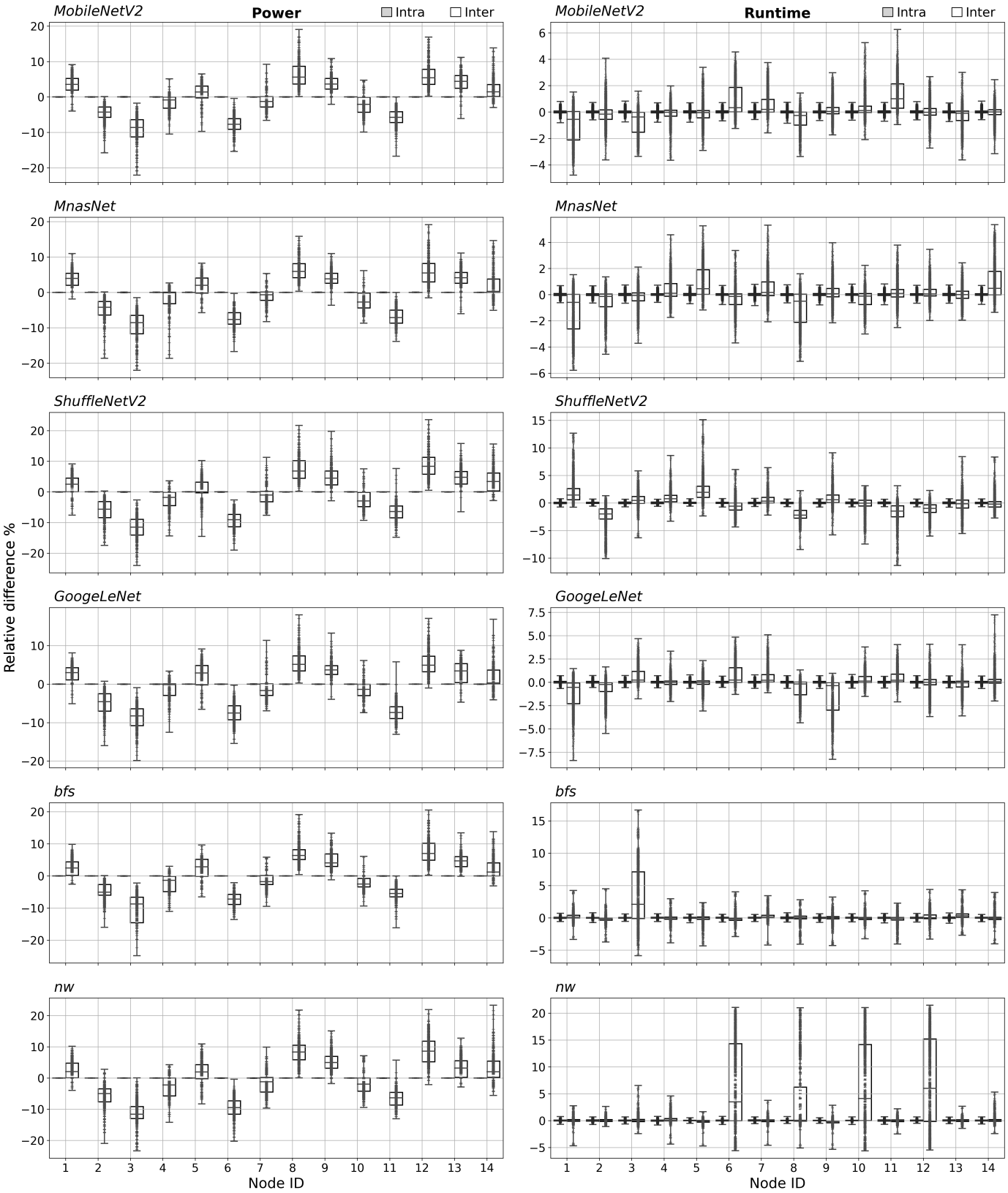

Fig. 5. Boxplots[3] presenting the distribution of rd (grey) and RD (white) values for power consumption (left) and runtime (right) for six benchmarks on the Jetson AGX. The x-axis shows the node ID (AGX board identifier), and the y-axis indicates the rd and RD values.

challenge of building accurate models (in the context of model-based runtime optimizations). We focus here only on runtime, since the high levels of variability we find for power (which are frequently larger than 30% and as high as 80%, if we consider the combined influence of inter-node variability and temperature) make, this point self-evident for power.

Compile-time optimizations. As a result of the stringent resource constraints in edge/embedded environments, it is desirable to optimize an application's binary for each target architecture [67]. While this increases the compilation time dramatically, the expectation is that this overhead is amortised over a large number of deployments [67], hence, a *compile once, deploy on many* approach is adopted. The underlying assumption is that the deployment nodes are identical to the one where the binary is tuned. Our experiments show that this assumption does not hold even under strict, lab-environment conditions that explicitly prioritize controlling variability: when all sources of variability we explore are taken into account, the observed effect is often beyond a 5% range, and, sometimes higher than 10%.

This magnitude matters for compile-time optimizations. An informative data point is provided by a survey by Mytcowicz et al. [5]: the authors report that the median speedup reported by 88 out of 133 papers surveyed that are published at top compiler and architecture conferences (e.g., ASPLOS, PACT, PLDI, and CGO) was $\approx 10\%$. *Given the magnitude of variability that we report in this study, without properly accounting for variability, the improvements reported by these studies could be lower.* To be clear: We do not argue these papers present incorrect conclusions; we argue that in an area like compile-time optimizations, where single digit performance gains are found worthwhile, since these gains are within the range of variability introduced by the various factors we characterize - inter-node, temperature, and even intra-node without variability controls - proper methodology and a good understanding of variability are necessary for meaningful conclusions.

The same study presents a second data point: the authors report *O3/O2* speed ups that are generally within the $\pm 5\%$ range (with few exceptions extending to the $\pm 10\%$). This is again a range that is well within the bounds we report on inter-node variability. This difference, if not accounted for in the methodology, one can reach a spurious conclusion about the benefits of one optimization level to another.

Model-based runtime optimization. In edge environments meeting QoS demands is a primary goal and two factors stand out: the application's performance (i.e., responsiveness, accuracy), and its energy consumption (i.e., battery life) [4]. Gaudette et al. [4] argue that building accurate performance and power models is essential to inform navigating this trade-off space.

Past work has experimented with a multitude of models, from those that brute-force the space (e.g., NeuOS [1] collects data at all configuration points and uses a lookup table), to those based on machine learning (e.g., Airavat [3] and PredJoule [2] employ a learning-based approach that relies on training models using data collected from a single board only). Variability matters during model building, and use.

Since generally a *model on one node, deploy everywhere* approach is generally taken, it is important to properly characterize the space and take various sources of variability (e.g., inter-node, and temperature) into account when building/seeding the model. Once a variability characterization is done, this will: *(i)* help understand the expected accuracy of the model when deployed in practice; and, possibly, *(ii)* lead to a decision to build models that take variability into account from the start (e.g., based on collecting data over multiple nodes, and/or taking temperature into account as a relevant factor). This will increase the cost of model development and may render some of the current techniques too costly (e.g., NeuOS [1] based on brute-forcing the space).

One of the few projects [4], [33], [68] that goes in this direction is Mirage [69]. On a hardware platform similar to ours, they show that by accounting for inter-node variability one can predict runtime and power consumption of DNN workloads at different frequency settings with up to 15.4% improvement over models that do not account for it.

In summary, characterizing variability is critical as it informs the developers about: *(i)* the lowest bound of variability that they should account for during runtime, and *(ii)* to what extent can each category of variability be mitigated. Without this characterization, wrong decisions during platform and/or applications' tuning could be made, leading to QoS violations.

VIII. Summary

We offer a methodology to characterize *intra-* and *inter-*node variability and apply it on two heterogeneous edge platforms: the NVIDIA Jetson AGX and Nano, for key characteristics: performance and power consumption. Firstly, we explore *intra-node* variability: highlight that it is unavoidable, understand to what degree deployment decisions can limit it, and offer a scale so that one can compare to what other studies report. Secondly, we characterize *inter-node* variability by answering two questions: *(i)* Are the platforms we study statistically different in terms of the applications' power draw and runtime performance? and *(ii)* What is the magnitude of these differences? Our results show that there is significant variability in power consumption across boards and moderate variability in runtime.

Acknowledgment

This project was sponsored in part by a gift fund from Huawei Toronto Heterogeneous Compiler Lab in Canada. We would like to extend our thanks to Shashwat Jaiswal for collecting additional data on the TX2 for the temperature variation experiment to increase confidence in our analysis. We also thank the reviewers of SEC '22 for their insightful feedback.

REFERENCES

[1] S. Bateni and C. Liu, "NeuOS: A Latency-Predictable Multi-Dimensional optimization framework for DNN-driven autonomous systems," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*. USENIX Association, Jul. 2020, pp. 371–385. [Online]. Available: https://www.usenix.org/conference/atc20/presentation/bateni

[2] S. Bateni, H. Zhou, Y. Zhu, and C. Liu, "Predjoule: A timing-predictable energy optimization framework for deep neural networks," in *2018 IEEE Real-Time Systems Symposium (RTSS)*, 2018, pp. 107–118.

[3] T. Baruah, Y. Sun, S. Dong, D. Kaeli, and N. Rubin, "Airavat: Improving energy efficiency of heterogeneous applications," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 731–736.

[4] B. G. et al., "Optimizing user satisfaction of mobile workloads subject to various sources of uncertainties," *IEEE Transactions on Mobile Computing*, vol. 18, no. 12, pp. 2941–2953, 2019.

[5] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Producing wrong data without doing anything obviously wrong!" *SIGARCH Comput. Archit. News*, vol. 37, no. 1, p. 265–276, mar 2009. [Online]. Available: https://doi.org/10.1145/2528521.1508275

[6] M. Li, M. Zhang, C. Wang, and M. Li, "Adatune: Adaptive tensor program compilation made efficient," *Advances in Neural Information Processing Systems*, vol. 33, pp. 14 807–14 819, 2020.

[7] A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree, "An empirical survey of performance and energy efficiency variation on intel processors," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, ser. E2SC'17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3149412.3149421

[8] B. Kocoloski and J. Lange, "Varbench: An experimental framework to measure and characterize performance variability," in *Proceedings of the 47th International Conference on Parallel Processing*, ser. ICPP 2018. New York, NY, USA: Association for Computing Machinery, 2018. [Online]. Available: https://doi.org/10.1145/3225058.3225125

[9] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807638

[10] T. Scogland, J. Azose, D. Rohr, S. Rivoire, N. Bates, and D. Hackenberg, "Node variability in large-scale power measurements: Perspectives from the green500, top500 and eehpcwg," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: Association for Computing Machinery, 2015. [Online]. Available: https://doi.org/10.1145/2807591.2807653

[11] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci, "Taming performance variability," in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. USA: USENIX Association, 2018, p. 409–425.

[12] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeyer, C. Maltzahn, R. Ricci, and A. Iosup, *Is Big Data Performance Reproducible in Modern Cloud Networks?* USA: USENIX Association, 2020, p. 513–528.

[13] P. Gupta, Y. Agarwal, L. Dolecek, N. Dutt, R. K. Gupta, R. Kumar, S. Mitra, A. Nicolau, T. S. Rosing, M. B. Srivastava, S. Swanson, and D. Sylvester, "Underdesigned and opportunistic computing in presence of hardware variability," *IEEE Transactions on Computer-Aided Design of integrated circuits and systems*, vol. 32, no. 1, pp. 8–23, 2012.

[14] D. Skinner and W. Kramer, "Understanding the causes of performance variability in hpc workloads," in *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.*, 2005, pp. 137–149.

[15] B. Balaji, J. McCullough, R. K. Gupta, and Y. Agarwal, "Accurate characterization of the variability in power consumption in modern mobile processors," in *2012 Workshop on Power-Aware Computing and Systems (HotPower 12)*. Hollywood, CA: USENIX Association, Oct. 2012. [Online]. Available: https://www.usenix.org/conference/hotpower12/workshop-program/presentation/Balaji

[16] R. Chandrashekhar, "Characterizing and leveraging processor variability in mobile devices for energy efficiency," Ph.D. dissertation, UC San Diego, 2013.

[17] A. Porterfield, R. Fowler, S. Bhalachandra, and W. Wang, "Openmp and mpi application energy measurement variation," in *Proceedings of the 1st International Workshop on Energy Efficient Supercomputing*, ser. E2SC '13. New York, NY, USA: Association for Computing Machinery, 2013. [Online]. Available: https://doi.org/10.1145/2536430.2536437

[18] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran, "Run-to-run variability on xeon phi based cray xc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3126908.3126926

[19] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, "Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1575–1588, 2018.

[20] Y. Wang, D. Nörtershäuser, S. Le Masson, and J.-M. Menaud, "Experimental characterization of variation in power consumption for processors of different generations," in *2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2019, pp. 702–710.

[21] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 44–54.

[22] F. W. Scholz and M. A. Stephens, "K-sample anderson–darling tests," *Journal of the American Statistical Association*, vol. 82, no. 399, pp. 918–924, 1987. [Online]. Available: https://doi.org/10.1080/01621459.1987.10478517

[23] J. Hodges, "The significance probability of the smirnov two-sample test," *Arkiv för Matematik*, vol. 3, no. 5, pp. 469–486, 1958.

[24] Y. Benjamini and Y. Hochberg, "Controlling the false discovery rate: A practical and powerful approach to multiple testing," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.2517-6161.1995.tb02031.x

[25] J. Algina, H. Keselman, and R. D. Penfield, "An alternative to cohen's standardized mean difference effect size: a robust parameter and confidence interval in the two independent groups case." *Psychological methods*, vol. 10, no. 3, p. 317, 2005.

[26] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 446–459.

[27] A. Das, S. Patterson, and M. Wittie, "Edgebench: Benchmarking edge computing platforms," in *2018 IEEE/ACM International Conference on Utility and Cloud Computing Companion (UCC Companion)*, 2018, pp. 175–180.

[28] W. Zhang, W. Wei, L. Xu, L. Jin, and C. Li, "Ai matrix: A deep learning benchmark for alibaba data centers," *arXiv preprint arXiv:1909.10562*, 2019.

[29] T. Hao, Y. Huang, X. Wen, W. Gao, F. Zhang, C. Zheng, L. Wang, H. Ye, K. Hwang, Z. Ren, and J. Zhan, "Edge aibench: Towards comprehensive end-to-end edge computing benchmarking," in *Benchmarking, Measuring, and Optimizing*, C. Zheng and J. Zhan, Eds. Cham: Springer International Publishing, 2019, pp. 23–30.

[30] P. Torelli and M. Bangale, "Measuring inference performance of machine-learning frameworks on edge-class devices with the mlmark benchmark," *Techincal Report. Available online: https://www. eembc. org/techlit/articles/MLMARK-WHITEPAPERFINAL-1. pdf (accessed on 5 April 2021)*, 2019.

[31] C.-J. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang,

B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine learning at facebook: Understanding inference at the edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[32] B. Gaudette, C.-J. Wu, and S. Vrudhula, "Improving smartphone user experience by balancing performance and energy with probabilistic qos guarantee," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 52–63.

[33] J. C. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. C. Snoeren, and R. K. Gupta, "Evaluating the effectiveness of model-based power characterization," in *USENIX Annual Technical Conf*, vol. 20, 2011.

[34] L. Wanner, C. Apte, R. Balani, P. Gupta, and M. Srivastava, "Hardware variability-aware duty cycling for embedded sensors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 6, pp. 1000–1012, 2013.

[35] S. Borkar, "Designing reliable systems from unreliable components: The challenges of transistor variability and degradation," *IEEE Micro*, vol. 25, no. 6, p. 10–16, nov 2005. [Online]. Available: https://doi.org/10.1109/MM.2005.110

[36] H. S. Gunawi, R. O. Suminto, R. Sears, C. Golliher, S. Sundararaman, X. Lin, T. Emami, W. Sheng, N. Bidokhti, C. McCaffrey, D. Srinivasan, B. Panda, A. Baptist, G. Grider, P. M. Fields, K. Harms, R. B. Ross, A. Jacobson, R. Ricci, K. Webb, P. Alvaro, H. B. Runesha, M. Hao, and H. Li, "Fail-slow at scale: Evidence of hardware performance faults in large production systems," *ACM Trans. Storage*, vol. 14, no. 3, oct 2018. [Online]. Available: https://doi.org/10.1145/3242086

[37] N. Dutt, P. Gupta, A. Nicolau, L. A. D. Bathen, and M. Gottscho, "Variability-aware memory management for nanoscale computing," in *2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2013, pp. 125–132.

[38] B. Rountree, D. K. Lowenthal, B. R. de Supinski, M. Schulz, V. W. Freeh, and T. Bletsch, "Adagio: Making dvs practical for complex hpc applications," in *Proceedings of the 23rd International Conference on Supercomputing*, ser. ICS '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 460–469. [Online]. Available: https://doi.org/10.1145/1542275.1542340

[39] J. D. Davis, S. Rivoire, M. Goldszmidt, and E. K. Ardestani, "Accounting for variability in large-scale cluster power models." Exascale Evaluation and Research Techniques Workshop (EXERT), 2011.

[40] T. Wilde, A. Auweter, H. Shoukourian, and A. Bode, "Taking advantage of node power variation in homogenous hpc systems to save energy," in *International Conference on High Performance Computing*. Springer, 2015, pp. 376–393.

[41] W. Lavrijsen, C. Iancu, W. de Jong, X. Chen, and K. Schwan, "Exploiting variability for energy optimization of parallel programs," in *Proceedings of the Eleventh European Conference on Computer Systems*, 2016, pp. 1–16.

[42] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda *et al.*, "Analyzing and mitigating the impact of manufacturing variability in power-constrained supercomputing," in *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2015, pp. 1–12.

[43] T. Scogland, J. Azose, D. Rohr, S. Rivoire, N. Bates, and D. Hackenberg, "Node variability in large-scale power measurements: perspectives from the green500, top500 and eehpcwg," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–11.

[44] A. Marathe, Y. Zhang, G. Blanks, N. Kumbhare, G. Abdulla, and B. Rountree, "An empirical survey of performance and energy efficiency variation on intel processors," in *Proceedings of the 5th International Workshop on Energy Efficient Supercomputing*, 2017, pp. 1–8.

[45] S. Chunduri, K. Harms, S. Parker, V. Morozov, S. Oshin, N. Cherukuri, and K. Kumaran, "Run-to-run variability on xeon phi based cray xc systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2017, pp. 1–13.

[46] F. Fraternali, A. Bartolini, C. Cavazzoni, and L. Benini, "Quantifying the impact of variability and heterogeneity on the energy efficiency for a next-generation ultra-green supercomputer," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 7, pp. 1575–1588, 2017.

[47] H. Weisbach, B. Gerofi, B. Kocoloski, H. Härtig, and Y. Ishikawa, "Hardware performance variation: A comparative study using lightweight kernels," in *High Performance Computing*. Cham: Springer International Publishing, 2018, pp. 246–265.

[48] M. J. Campbell and M. J. Gardner, "Statistics in medicine: Calculating confidence intervals for some non-parametric analyses," *British medical journal (Clinical research ed.)*, vol. 296, no. 6634, p. 1454, 1988.

[49] *Jetson AGX Xavier Series: Thermal Design Guide*, NVIDIA, 9 2020. [Online]. Available: https://tinyurl.com/r7zeehya

[50] C. Leys, C. Ley, O. Klein, P. Bernard, and L. Licata, "Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median," *Journal of Experimental Social Psychology*, vol. 49, no. 4, pp. 764–766, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022103113000668

[51] D. G. Bonett, "Confidence interval for a coefficient of quartile variation," *Computational Statistics & Data Analysis*, vol. 50, no. 11, pp. 2953–2957, 2006. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947305001271

[52] T. Zhou and L. Zhu, "Conditional median absolute deviation," *Journal of Statistical Computation and Simulation*, vol. 85, no. 10, pp. 2101–2114, 2015. [Online]. Available: https://doi.org/10.1080/00949655.2014.922185

[53] B. Altunkaynak and H. Gamgam, "Bootstrap confidence intervals for the coefficient of quartile variation," *Communications in Statistics - Simulation and Computation*, vol. 48, no. 7, pp. 2138–2146, 2019. [Online]. Available: https://doi.org/10.1080/03610918.2018.1435800

[54] N. M. Razali, Y. B. Wah *et al.*, "Power comparisons of shapiro-wilk, kolmogorov-smirnov, lilliefors and anderson-darling tests," *Journal of statistical modeling and analytics*, vol. 2, no. 1, pp. 21–33, 2011.

[55] J. Li, "Effect size measures in a two-independent-samples case with nonnormal and nonhomogeneous data," *Behavior research methods*, vol. 48, no. 4, pp. 1560–1574, 2016.

[56] J. Cohen, *Statistical power analysis for the behavioral sciences*. A. Press, 2013.

[57] T. J. P. Team. (2022) NVIDIA Jetson AGX Xavier Specifications. NVIDIA. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-agx-xavier/

[58] NVIDIA. (2022) NVIDIA Jetson Nano Specifications. NVIDIA. [Online]. Available: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

[59] M. G. S. Murshed, C. Murphy, D. Hou, N. Khan, G. Ananthanarayanan, and F. Hussain, "Machine learning at the network edge: A survey," *ACM Comput. Surv.*, vol. 54, no. 8, oct 2021. [Online]. Available: https://doi.org/10.1145/3469029

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[61] S. Marcel and Y. Rodriguez, "Torchvision the machine-vision package of torch," in *Proceedings of the 18th ACM International Conference on Multimedia*, ser. MM '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 1485–1488. [Online]. Available: https://doi.org/10.1145/1873951.1874254

[62] The PyTorch Team. (2021) CUDA Timing Events. NVIDIA. [Online]. Available: https://pytorch.org/docs/stable/cuda.html#torch.cuda.Event

[63] *INA3221 Power Monitors*, Texas Instruments, 3 2016. [Online]. Available: https://www.ti.com/product/INA3221

[64] P. M. Store. (2022) Watts Up Pro Portable Power Meter. Power Meter Store. [Online]. Available: https://www.powermeterstore.com/p1206/watts_up_pro.php

[65] C. Zhuo, S. Luo, H. Gan, J. Hu, and Z. Shi, "Noise-aware dvfs for efficient transitions on battery-powered iot devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 7, pp. 1498–1510, 2020.

[66] A. E. Systems. (2022) Thermal Chamber Configurations. Associated Environmental Systems. [Online]. Available: https://www.associatedenvironmentalsystems.com/products/sd-508

[67] T. Kisuki, P. Knijnenburg, M. O'Boyle, and H. Wijshoff, "Iterative compilation in program optimization," in *Proc. CPC'10 (Compilers for Parallel Computers)*. Citeseer, 2000, pp. 35–44.

[68] Y. G. Kim and C.-J. Wu, "Autoscale: Energy efficiency optimization for stochastic edge inference using reinforcement learning," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1082–1096.

[69] H. A. Abdelhafez, H. Halawa, M. O. Ahmed, K. Pattabiraman, and M. Ripeanu, "Mirage: Machine learning-based modeling of identical replicas of the jetson agx embedded platform," in *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, 2021, pp. 26–40.