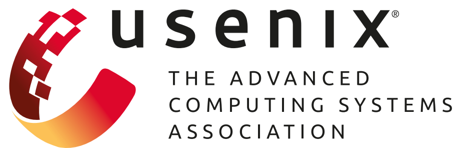


ALERT: Accurate Learning for Energy and Timeliness

Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu, University of Chicago

https://www.usenix.org/conference/atc20/presentation/wan

This paper is included in the Proceedings of the 2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the 2020 USENIX Annual Technical Conference is sponsored by USENIX.

ALERT: Accurate Learning for Energy and Timeliness

Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, Shan Lu
The University of Chicago

Abstract

An increasing number of software applications incorporate runtime Deep Neural Networks (DNNs) to process sensor data and return inference results to humans. Effective deployment of DNNs in these interactive scenarios requires meeting latency and accuracy constraints while minimizing energy, a problem exacerbated by common system dynamics.

Prior approaches handle dynamics through either (1) system-oblivious DNN adaptation, which adjusts DNN latency/accuracy tradeoffs, or (2) application-oblivious system adaptation, which adjusts resources to change latency/energy tradeoffs. In contrast, this paper improves on the state-of-the-art by coordinating application- and system-level adaptation. ALERT, our runtime scheduler, uses a probabilistic model to detect environmental volatility and then simultaneously select both a DNN and a system resource configuration to meet latency, accuracy, and energy constraints. We evaluate ALERT on CPU and GPU platforms for image and speech tasks in dynamic environments. ALERT's holistic approach achieves more than 13% energy reduction, and 27% error reduction over prior approaches that adapt solely at the application or system level. Furthermore, ALERT incurs only 3% more energy consumption and 2% higher DNN-inference error than an oracle scheme with perfect application and system knowledge.

1 Introduction

1.1 Motivation

Deep neural networks (DNNs) have become a key workload for many computing systems due to their high inference accuracy. This accuracy, however, comes at a cost of long latency, high energy usage, or both. Successful DNN deployment requires meeting a variety of user-defined, application-specific goals for latency, accuracy, and often energy in unpredictable, dynamic environments.

Latency constraints naturally arise with DNN deployments when inference interacts with the real world as a consumer—processing data streamed from a sensor—or a producer—returning a series of answers to a human. For example, in motion tracking, a frame must be processed at camera speed [40]; in simultaneous interpretation, translation must be provided every 2–4 seconds [56]. Violating these deadlines may lead to severe consequences: if a self-driving vehicle cannot act within a small time budget, life threatening accidents could follow [53].

Accuracy and energy requirements are also common and may vary for different applications in different operating environments. On one hand, low inference accuracy can lead to software failures [67, 80]. On the other hand, it is beneficial to minimize DNN energy or resource usage to extend mobile-battery time or reduce server-operation cost [41].

These requirements are also highly dynamic. For example, the latency requirement for a job could vary dynamically depending on how much time has already been consumed by related jobs before it [53]; the power budget and the accuracy requirement for a job may switch among different settings depending on what type of events are currently sensed [1]. Additionally, the latency requirement may change based on the computing system's current context; e.g., in robotic vision systems the latency requirement can change based on the robot's latency and distance from perceived pedestrians [18].

Satisfying all these requirements in a dynamic computing environment where the inference job may compete for resources against unpredictable, co-located jobs is challenging. Although prior work addresses these problems at either the application level or system level separately, each approach by itself lacks critical information that could be used to produce better results.

At the application level, different DNN designs—with different depths, widths, and numeric precisions—provide various latency-accuracy trade-offs for the same inference task [26, 39, 42, 77, 85]. Even more dynamic schemes have been proposed that adapt the DNN by dynamically changing its structure at the beginning of [22, 61, 84, 89] or during [34, 35, 49, 52, 82, 86, 88] every inference tasks.

Although helpful, these techniques are sub-optimal

without considering system-level adaptation options. For example, under energy pressure, these application-level adaptation techniques have to switch to lower-accuracy DNNs, sacrificing accuracy for energy saving, even if the energy goal could have been achieved by lowering the system power setting (if there is sufficient latency budget).

At the system level, machine learning [4, 14, 15, 51, 63, 68, 69, 79] and control theory [32, 37, 44, 45, 62, 70, 74, 93] based techniques have been proposed to dynamically assign system resources to better satisfy system and application constraints.

Unfortunately, without considering the option of application adaptions, these techniques also reach sub-optimal solutions. For example, when the current DNN offers much higher accuracy than necessary, switching to a lower-precision DNN may offer much more energy saving than any system-level adaptation techniques. This problem is exacerbated because, in the DNN design space, very small drops in accuracy enable dramatic reductions in latency, and therefore system resource requirements.

A cross-stack solution would enable DNN applications to meet multiple, dynamic constraints. However, offering such a holistic solution is non-trivial. The combination of DNN and system-resource adaptation creates a huge configuration space, making it difficult to dynamically and efficiently predict which combination of DNN and system settings will meet all the requirements optimally. Furthermore, without careful coordination, adaptations at the application and system level may conflict and cause constraint violations, like missing a latency deadline due to switching to higher-accuracy DNN and lower power setting at the same time.

1.2 Contributions

This paper presents ALERT, a cross-stack runtime system for DNN inference to meet user goals by simultaneously adapting both DNN models and system-resource settings.

Understanding the challenges We profile DNN inference across applications, inputs, hardware, and resource contention confirming there is a high variation in inference time. This leads to challenges in meeting not only latency but also energy and accuracy requirements. Furthermore, our profiling of 42 existing DNNs for image classification confirms that different designs offer a wide spectrum of latency, energy, and accuracy tradeoffs. In general, higher accuracy comes at the cost of longer latency and/or higher energy consumption. These trade-offs offered provide both opportunities and challenges to holistic inference management (Section 2).

Run-time inference management We design ALERT, a DNN inference management system that dynamically selects and adapts a DNN and a system-resource setting together to handle changing system environments and meet dynamic energy, latency, and accuracy requirements[1] (Section 3).

[1] ALERT provides probabilistic, not hard guarantees, as the latter requires much more conservative configurations, often hurting both energy and accuracy. Section 3.6 discusses this issue further.

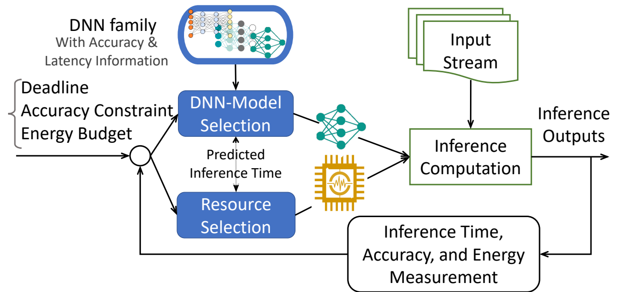


Figure 1: ALERT inference system

ALERT is a feedback-based run-time. It measures inference accuracy, latency, and energy consumption; it checks whether the requirements on these goals are met; and, it then outputs both system and application-level configurations adjusted to the current requirements and operating conditions. ALERT focuses on meeting constraints in *any* two dimensions while optimizing the third; e.g., minimizing energy given accuracy and latency requirements or maximizing accuracy given latency and energy budgets.

The key is estimating how DNN and system configurations interact to affect the goals. To do so, ALERT addresses three primary challenges: (1) the combined DNN and system configuration space is huge, (2) the environment may change dynamically (including input, available resources, and even the required constraints), and (3) the predictions must be low overhead to have negligible impact on the inference itself.

ALERT addresses these challenges with a *global slow-down factor*, a random variable relating the current runtime environment to a nominal profiling environment. After each inference task, ALERT estimates the global slow-down factor using a Kalman filter. The global slow-down factor's mean represents the expected change compared to the profile, while the variance represents the current volatility. The mean provides a single scalar that modifies the predicted latency/accuracy/energy for *every* DNN/system configuration—a simple mechanism that leverages commonality among DNN architectures to allow prediction for even rarely used configurations (tackle challenge-1), while incorporating variance into predictions naturally makes ALERT conservative in volatile environments and aggressive in quiescent ones (tackle challenge-2). The global slow-down factor and Kalman filter are efficient to implement and low-overhead (tackle challenge-3). Thus, ALERT combines the global slow-down factor with latency, power, and accuracy measurements to select the DNN and system configuration with the highest likelihood of meeting the constraints optimally.

We evaluate ALERT using various DNNs and application domains on different (CPU and GPU) machines under various constraints. Our evaluation shows that ALERT overcomes dynamic variability efficiently. Across various experimental

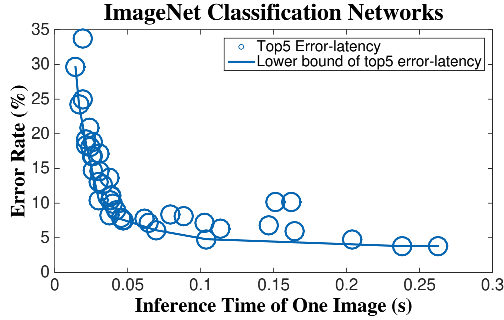


Figure 2: Tradeoffs for 42 DNNs (CPU2).

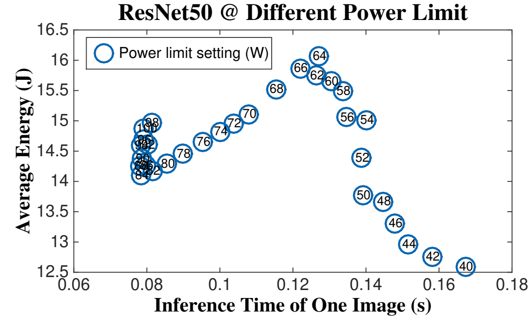


Figure 3: Tradeoffs for ResNet50 at different power settings (CPU2). (Numbers inside circles are power limit settings.)

settings, ALERT meets constraints while achieving within 93–99% of optimal energy saving or accuracy optimization. Compared to approaches that adapt at application-level or system-level only ALERT achieves more than 13% energy reduction, and 27% error reduction (Section 5).

2 Understanding Deployment Challenges

We conduct an empirical study to examine the large trade-off space offered by different DNN designs and system settings (Sec. 2.1), and the timing variability of inference (Sec. 2.2).

	Embedded	CPU1	CPU2	GPU
CPU	ARM Cortex A-15 @2.0 GHz	Core-i7 @2.2 GHz	Xeon(R) Gold 6126 @2.60GHz	Core-i7 @2.2 GHz
GPU	none	none	none	RTX 2080
Memory	DDR3 2G	DDR4 16G	DDR4 16G*12	DDR4 16G
LLC	2MB	9MB	19.25MB	9MB

Table 1: Hardware platforms used in our experiments

ID	Task	DNN Models	Datasets
IMG1	Image Classification	VGG16 [78]	ILSVRC2012 (ImageNet)
IMG2		ResNet50 [29]	
NLP1	Sentence Prediction	RNN	Penn Treebank [59]
NLP2	Question Answering	Bert [17]	Stanford Q&A Dataset (SQuAD) [71]

Table 2: ML tasks and benchmark datasets in our experiments

We use two canonical machine learning tasks, with state-of-the-art networks and common data-sets (see Table 2) on a diverse set of hardware platforms, representing embedded systems, laptops (CPU1), CPU servers (CPU2), and GPU platforms (see Table 1). The two tasks, image classification and natural language processing (NLP), are often deployed with deadlines—e.g., for motion tracking [40] and simultaneous interpretation [56]—and both have received wide attention leading to a diverse set of DNN models.

2.1 Understanding the Tradeoffs

Tradeoffs from DNNs We run **all** 42 image classification models provided by the Tensorflow website [76] on the 50000 images from ImageNet [16], and measure their average latency, accuracy (error rate), and energy consumption. The results from CPU2 are shown in Figure 2. We can clearly see two trends from the figure, which hold on other machines.

First, different DNN models offer a *wide* spectrum of accuracy (error rate in figure), latency, and energy. As shown in the figure, the fastest model runs almost 18× faster than the slowest one and the most accurate model has about 7.8× lower error rate than the least accurate. These models also consume a wide range—more than 20×—of energy usage.

Second, there is no magic DNN that offers both the best accuracy and the lowest latency, confirming the intuition that there exists a tradeoff between DNN accuracy and resource usage. Of course, some DNNs offer better tradeoffs than others. In Figure 2, all the networks sitting above the lower-convex-hull curve represent sub-optimal tradeoffs.

Tradeoffs from system settings We run ResNet50 under 31 power settings from 40–100W on CPU2. We consider a sensor processing scenario with periodic inputs, setting the period to the latency under 40W cap. We then plot the average energy consumed for the whole period (run-time plus idle energy) and the average inference latency in Figure 3.

The results reflect two trends, which hold on other machines. First, a large latency/energy space is available by changing system settings. The fastest setting (100W) is more than 2× faster than the slowest setting (40W). The most energy-hungry setting (64W) uses 1.3× more energy than the least (40W). Second, there is no easy way to choose the best setting. For example, 40W offers the lowest energy, but highest latency. Furthermore, most of these points are sub-optimal in terms of energy and latency tradeoffs. For example, 84W should be chosen for extremely low latency deadlines, but all other nearby points (from 52–100) will harm latency, energy or both. Additionally, when deadlines change or when there is resource contention, the energy-latency curve also changes and different points become optimal.

Summary: DNN models and system-resource settings offer a huge trade-off space. The energy/latency tradeoff space is not smooth (when accounting for deadlines and idle power) and optimal operating points cannot be found with simple gradient-based heuristics. Thus, there is a great

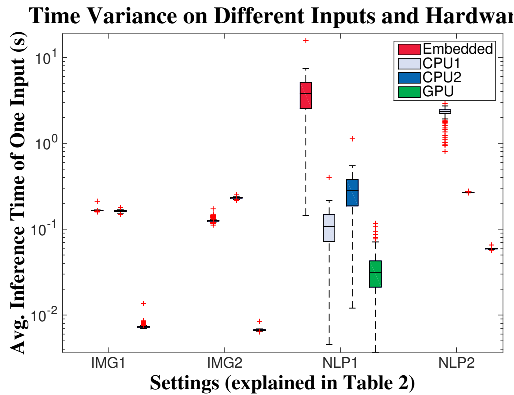


Figure 4: Latency variance across inputs for different tasks and hardware (Most tasks have 3 boxplots for 3 hardware platforms, CPU1-2, GPU from left to right; NLP1 has an extra boxplot for Embedded; other tasks run out of memory on Embedded; every box shows the 25th–75th percentile; points beyond the whiskers are >90th or <10th).

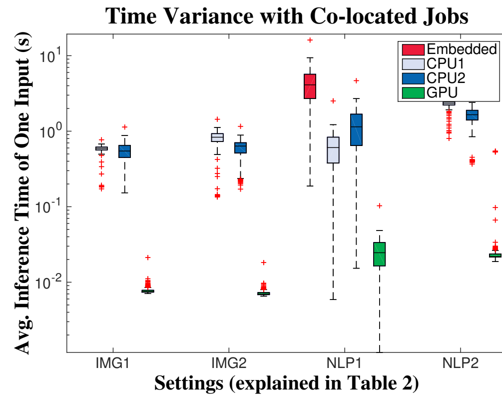


Figure 5: Latency variance with co-located jobs (the memory-intensive STREAM benchmark [60] co-located on Embedded, CPU1-2; GPU-intensive Backprop [8] co-located on GPU)

opportunity and also a great challenge in picking different DNN models and system-resource settings to satisfy inference latency, accuracy, and energy requirements.

2.2 Understanding Variability

To understand how DNN-inference varies across inputs, platforms, and run-time environment and hence how (not) helpful is off-line profiling, we run a set of experiments below, where we feed the network one input at a time and use 1/10 of the total data for warm up, to emulate real-world scenarios. We plot the inference latency without and with co-located jobs in Figure 4 and 5, and we see several trends.

First, deadline violation is a realistic concern. Image classification on video has deadlines ranging from 1 second to the camera latency (e.g., 1/60 seconds) [40]; the two NLP tasks, have deadlines around 1 second [64]. There is clearly no single inference task that meets all deadlines on all hardware.

Second, the inference variation among inputs is relatively small particularly when there are no co-located jobs (Fig. 4), except for that in NLP1, where this large variance is mainly caused by different input lengths. For other tasks, outlier inputs exist but are rare.

Third, the latency and its variation across inputs are both greatly affected by resource contention. Comparing Figure 5 with Figure 4, we can see that the co-located job has increased both the median latency, the tail inference, and the difference between these two for all tasks on all platforms. This trend also applies to other contention cases.

While the discussion above is about latency, similar conclusions apply to inference accuracy and energy: the accuracy typically drops to close to 0 when the inference time exceeds the latency requirement, and the energy consumption naturally changes with inference time.

Summary: Deadline violations are realistic concerns and inference latency varies greatly across platforms, under contention, and sometimes across inputs. Clearly, sticking to one static DNN design across platforms and workloads leads to an unpleasant trade-off: always meeting the deadline by sacrificing accuracy or energy in most settings, or achieving a high accuracy some times but exceeding the deadline in others. Furthermore, it is also sub-optimal to make run-time decisions based solely on off-line profiling, considering the variation caused by run-time contention.

2.3 Understanding Potential Solutions

We now show how confining adaptation to a single layer (just application or system) is insufficient. We run the ImageNet classification on *CPU1*. We examine a range of latency (0.1s-0.7s) and accuracy constraints (85%-95%), and try meeting those constraints while minimizing energy by either (1) configuring just the DNN (selecting a DNN from a family, like that in Figure 2) or (2) configuring just the system (by selecting resources to control energy–latency tradeoffs as in Figure 3). We compare these single-layer approaches to one that simultaneously picks the DNN and system configuration. As we are concerned with the ideal case, we create oracles by running 90 inputs in all possible DNN and system configurations, from which we find the best configuration for each input. The App-level oracle uses the default system setting. The Sys-level oracle uses the default (highest accuracy) DNN.

Figure 6 shows the results. As we have a three dimensional problem—meeting accuracy and latency constraints with minimal energy—we linearize the constraints and show them on the x-axis (accuracy is faster changing, with latency slower, so each latency bin contains all accuracy goals). There are several important conclusions here. First, the App-only approach meets all possible accuracy and latency constraints, while the Sys-only approach cannot meet any constraints below 0.3s. Second, across the entire constraint range, App-

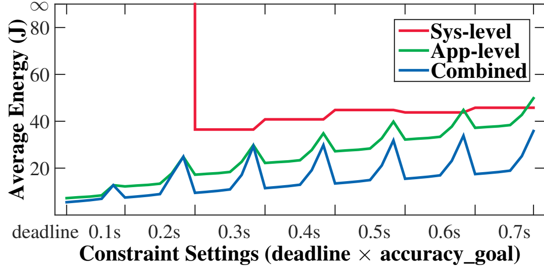


Figure 6: Minimize energy task with latency and accuracy constraint @ CPU1. (∞ means unable to meet the constraints)

only consumes significantly more energy than Combined (60% more on average). The intuition behind Combined's superiority is that there are discrete choices for DNNs; so when one is selected, there are almost always energy saving opportunities by tailoring resource usage to that DNN's needs.

Summary: Combining DNN and system level approaches achieves better outcomes. If left solely to the application, energy will be wasted. If left solely to the system, many achievable constraints will not be met.

3 ALERT Run-time Inference Management

ALERT's runtime system navigates the large tradeoff space created by *combining* DNN-level and system-level adaptation. ALERT meets user-specified latency, accuracy, and energy constraints and optimization goals while accounting for run-time variations in environment or the goals themselves.

3.1 Inputs & Outputs of ALERT

ALERT's inputs are specifications about (1) the adaption options, including a set of DNN models $\mathbb{D} = \{d_i \mid i = 1 \cdots K\}$ and a set of system-resource settings, expressed as different power-caps $\mathbb{P} = \{P_j \mid j = 1 \cdots L\}$; and (2) the user-specified requirements on latency, accuracy, and energy usage, which can take the form of meeting constraints in any two of these three dimensions while optimizing the third. ALERT's output is the DNN model $d_i \in \mathbb{D}$ and the system-resource setting $p_j \in \mathbb{P}$ for the next inference-task input.

Formally, ALERT selects a DNN d_i and a system-resource setting p_j to fulfill *either* of these user-specified goals.

Maximizing inference accuracy q (minimizing error) for an energy budget $\mathbf{E}_{\text{goal}}$ and inference deadline $\mathbf{T}_{\text{goal}}$:

$$\arg\max_{i,j} q_{i,j} \quad \text{s.t. } e_{i,j} \leq \mathbf{E}_{\text{goal}} \wedge t_{i,j} \leq \mathbf{T}_{\text{goal}} \tag{1}$$

Minimizing the energy use e for an accuracy goal $\mathbf{Q}_{\text{goal}}$ and inference deadline $\mathbf{T}_{\text{goal}}$:

$$\arg\min_{i,j} e_{i,j} \quad \text{s.t. } q_{i,j} \geq \mathbf{Q}_{\text{goal}} \wedge t_{i,j} \leq \mathbf{T}_{goal} \tag{2}$$

We omit the discussion of meeting energy and accuracy constraints while minimizing latency as it is a trivial extension of the discussed techniques and we believe it to be the least practically useful. We also omit the problem of optimizing all three dimensions, as it creates a feasibility problem, leaving nothing for optimization—lowest latency and highest accuracy are impractical to achieve simultaneously.

Generality Along the DNN-adaptation side, the input DNN set can consist of any DNNs that offer different accuracy, latency, and energy tradeoffs; e.g., those in Figure 3. In particular, ALERT can work with either or both of the broad classes of DNN adaptation approaches that have arisen recently, including: (1) traditional DNNs where the adaptation option should be selected prior to starting an inference task [20,22,61,84,89] and (2) anytime DNNs that produce a series of outputs as they execute [34,35,49,52,82,86,88]. These two classes are similar in that they both vary things like the network depth or width to create latency/accuracy tradeoffs.

On the system-resource side, ALERT uses a *power cap* as the proxy to system resource usage. Since both hardware [13] and software resource managers [33,72,90] can convert power budgets into optimal performance resource allocations, ALERT is compatible with many different schemes from both commercial products and the research literature.

3.2 ALERT Workflow

ALERT works as a feedback controller. It follows four steps to pick the DNN and resource settings for each input n:

1) Measurement. ALERT records the processing time, energy usage, and computes inference accuracy for $n-1$.

2) Goal adjustment. ALERT updates the time goal T_{goal} if necessary, considering the potential latency-requirement variation across inputs. In some inference tasks, a set of inputs share one combined requirement (e.g., in the NLP1 task in Table 2, all the words in a sentence are processed by a DNN one by one and share one sentence-wise deadline) and hence delays in previous input processing could greatly shorten the available time for the next input [1,47]. Additionally, ALERT sets the goal latency to compensate for its own, worst-case overhead so that ALERT itself will not cause violations.

3) Feedback-based estimation. ALERT computes the expected latency, accuracy, and energy consumption for every combination of DNN model and power setting.

4) Picking a configuration. ALERT feeds all the updated estimations of latency, accuracy, and energy into Eqs. 1 and 2, and gets the desired DNN model and power-cap setting for n.

The key task is step 3: the estimation needs to be accurate and fast. In the remainder of this section, we discuss key ideas and the exact algorithm of our feedback-based estimation.

3.3 Key Ideas of ALERT Estimation

Strawman Solving Eqs. 1 and 2 would be trivially easy if the deployment environment is guaranteed to match the training and profiling environment: we could estimate $t_{i,j}$ to be the average (or worst case, etc) inference time $t_{i,j}^{\text{prof}}$ over a set of profiling inputs under model d_i and power setting p_j. However, this approach does not work given the dynamic input, contention, and requirement variation.

Next, we present the key ideas behind how ALERT estimates the inference latency, accuracy, and energy consumption under model d_i and power setting p_j.

How to estimate the inference latency $t_{i,j}$? To handle the run-time variation, a potential solution is to apply an estimator, like a Kalman filter [55], to make dynamic predictions based on recent history about inferences under model d_i and power p_j. The problem is that most models and power settings will not have been picked recently and hence would have no recent history to feed into the estimator. This problem is a direct example of the challenge imposed by the large space of combined application and system options.

Idea 1: Handle the large selection space with a single scalar value. To make effective online estimation for *all* combinations of models and power settings, ALERT introduces a *global slow-down factor* ξ to capture how the current environment differs from the profiled environment (e.g., due to co-running processes, input variation, or other changes). Such an environmental slow-down factor is independent from individual model or power selection. It can fully leverage execution history, no matter which models and power settings were recently used; it can then be used to estimate $t_{i,j}$ based on $t_{i,j}^{\text{prof}}$ for all d_i and p_j combinations.

Applying a *global* slowdown factor for *all* combinations of application and system-level settings is crucial for ALERT to make quick decisions for every inference task. Although it is possible that some perturbations may lead to different slowdowns for different configurations, the slight loss of accuracy here is out-weighed by the benefit of having a simple mechanism that allows prediction even for configurations that have not been used recently.

This idea is also novel for ALERT, as previous cross-stack management systems all use much more complicated models to estimate and select different setting combinations (e.g., using model predictive control to estimate combinations of settings [57]). ALERT's global slowdown factor is based on several unique features of DNN families that accomplish the same task with different accurarcy/latency tradeoffs. We categorize these features as: (1) similarity of code paths and (2) proportionality of structure. The first is based on the observation that DNNs do not have complex conditional code dependences, so we do not need to worry about the case where different inputs would exercise very different code paths. Thus, what ALERT learns about latency, accuracy, and energy for one input will always inform it about future inputs. The second feature refers to the fact that as DNNs in a family scale in latency, the proportion of different operations tend to be similar, so what ALERT learns about one DNN in the family generally applies to other DNNs in the same family. These properties of DNNs do not hold for many other types of software, where different inputs or additional functionality can invoke entirely different code paths, with different resource requirements or responses.

How to estimate the accuracy under a deadline? Given a deadline $\mathbf{T}_{\text{goal}}$, the inference accuracy delivered by model d_i and power setting p_j is determined by three factors, as shown in Eq. 3: (1) whether the inference result, which takes time $t_{i,j}$, can be generated before the deadline $\mathbf{T}_{\text{goal}}$; (2) if yes, the accuracy is determined by the model d_i;[2] (3) if not, the accuracy drops to that offered by a backup result q_{fail}. For traditional DNN models, without any output at the deadline, a random guess will be used and q_{fail} will be much worse than q_i. For anytime DNN models that output multiple results as they are ready, the backup result is the latest output [34, 35, 49, 52, 82, 86, 88], which we discuss more in Section 3.5.

$$q_{i,j}[\mathbf{T}_{\text{goal}}] = \begin{cases} q_i & \text{, if } t_{i,j} \leq \mathbf{T}_{\text{goal}} \\ q_{\text{fail}} & \text{, otherwise} \end{cases} \tag{3}$$

A potential solution to estimate accuracy $q_{i,j}$ at the deadline $\mathbf{T}_{\text{goal}}$ is to simply feed the estimated $t_{i,j}$ into Eq. 3. However, this simple approach fails to account for two issues. First, while DNNs are generally well-behaved, significant tail effects are possible (see Figure 4). Second, Eq. 3 is not linear, and is best understood as a step function, where a failure to complete inference by the deadline results in a worthless inference output (q_{fail}). Combined, these two issues mean that for tail inputs, inference will produce a worthless result; i.e., accuracy is not proportional to latency, but can easily fall to zero for tail inputs. The tail will, of course, be increased if there is any unexpected resource contention. Therefore, the simple approach of using the mean latency prediction fails to account for the non-linear affects of latency on accuracy.

Idea 2: handle the runtime variation and account for tail behavior To handle the run-time variability mentioned in Section 1, ALERT treats the execution time $t_{i,j}$ and the global slow-down factor ξ as *random variables* drawn from a normal distribution. ALERT uses a recently proposed extension to the Kalman filter to adaptively update the noise covariance [2]. While this extension was originally proposed to produce better estimates of the mean, a novel approach in ALERT is using this covariance estimate as a measure of system volatility. ALERT uses this Kalman filter extension to predict not just the mean accuracy, but also the likelihood of meeting the accuracy requirements in the current operating environment. Section 5.3 shows the advantages of our extensions.

[2] Since it could be infeasible to calculate the exact inference accuracy at run time, ALERT uses the average training accuracy of the selected DNN model d_i, denoted as q_i, as the inference accuracy, as long as the inference computation finishes before the specified deadline.

How to minimize energy or satisfy energy constraints? Minimizing energy or satisfying energy constraints is complicated, as the energy is related to, but cannot be easily calculated by, the complexity of the selected model d_i and the power cap p_j. As discussed in Section 2.2, the energy consumption includes both that used during the inference under a given model d_i and that used during the inference-idle period, waiting for the next input. Consequently, it is not straightforward to decide which power setting to use.

Idea 3. ALERT leverages insights from previous research, which shows that energy for latency-constrained systems can be efficiently expressed as a mathematical optimization problem [7,48,50,62]. These frameworks optimize energy by scheduling available configurations in time. Time is assigned to configurations so that the average performance hits the desired latency target and the overall energy (including idle energy) is minimal. The key is that while the configuration space is large, the number of constraints is small (typically just two). Thus, the number of configurations assigned a non-zero time is also small (equal to the number of constraints) [48]. Given this structure, the optimization problem can be solved using a binary search over available configurations, or even more efficiently with a hash table [62].

The only difficulty applying prior work to ALERT is that prior work assumed there was only a single job running at a time, while ALERT assumes that other applications might contend for resources. Thus, ALERT cannot assume that there is a single system-idle state that will be used whenever the DNN is not executing. To address this challenge, ALERT continually estimates the system power when DNN inference is idle (but other non-inference tasks might be active), $p_{DNNidle}$, transforming Eq. 1 is transformed into:

$$\arg\max_{i,j} q_{i,j}[\mathbf{T}_{\text{goal}}] \quad \text{s.t. } p_{i,j} \cdot t_{i,j} + p_{DNNidle} \cdot t_{DNNidle} \leq \mathbf{E}_{\text{goal}} \tag{4}$$

3.4 ALERT Estimation Algorithm

Global Slow-down Factor ξ. As discussed in Idea-1, ALERT uses ξ to reflect how the run-time environment differs from the profiling environment. Conceptually, if the inference task under model d_i and power-cap p_j took time $t_{i,j}$ at run time and took $t_{i,j}^{\text{prof}}$ on average to finish during profiling, the corresponding ξ would be $t_{i,j}/t_{i,j}^{prof}$. ALERT estimates ξ using recent execution history under any model or power setting.

Specifically, after an input $n-1$, ALERT computes $\xi^{(n-1)}$ as the ratio of the observed time $t_{i,j}^{(n-1)}$ to the profiled time $t_{i,j}^{\text{prof}}$, and then uses a Kalman Filter[3] to estimate the mean $\mu^{(n)}$ and variance $(\sigma^{(n)})^2$ of $\xi^{(n)}$ at input n. ALERT's formulation is defined in Eq. 5, where $K^{(n)}$ is the Kalman gain variable; R is a constant reflecting the measurement noise; $Q^{(n)}$ is the process noise capped with $Q^{(0)}$. We set a forgetting factor of process variance $\alpha = 0.3$ [2]. ALERT initially sets $K^{(0)} = 0.5$, $R = 0.001$, $Q^{(0)} = 0.1$, $\mu^{(0)} = 1$, $(\sigma^{(0)})^2 = 0.1$, following the standard convention [55].

$$\begin{cases} Q^{(n)} = \max\{Q^{(0)}, \alpha Q^{(n-1)} + (1\text{-}\alpha)(K^{(n-1)} y^{(n-1)})^2\} \\ K^{(n)} = \dfrac{(1-K^{(n-1)})(\sigma^{(n-1)})^2 + Q^{(n)}}{(1-K^{(n-1)})(\sigma^{(n-1)})^2 + Q^{(n)} + R} \\ y^{(n)} = t_{i,j}^{(n-1)}/t_{i,j}^{\text{prof}} - \mu^{(n-1)} \\ \mu^{(n)} = \mu^{(n-1)} + K^{(n)} y^{(n)} \\ (\sigma^{(n)})^2 = (1-K^{(n-1)})(\sigma^{(n-1)})^2 + Q^{(n)} \end{cases} \tag{5}$$

Then, using $\xi^{(n)}$, ALERT estimates the inference time of input n under any model d_i and power cap p_j: $t_{i,j}^{(n)} = \xi^{(n)} * t_{i,j}^{\text{prof}}$.

Probability of meeting the deadline. Given the Kalman Filter estimation for the global slowdown factor, we can calculate $Pr_{i,j}$, the probability that the inference completes before the deadline T_{goal}. ALERT computes this value using a cumulative distribution function (CDF) based on the normal distribution of $\xi^{(n)}$ estimated by the Kalman Filter:

$$\begin{aligned} Pr_{i,j} &= Pr[\xi^{(n)} \cdot t_{i,j}^{\text{prof}} \leq T_{goal}] = CDF(\xi^{(n)} \cdot t_{i,j}^{\text{prof}}, T_{goal}) \\ &= CDF(\mu^{(n)} \cdot t_{i,j}^{\text{prof}}, \sigma^{(n)}, T_{goal}) \end{aligned} \tag{6}$$

Accuracy. As discussed in Idea-2, ALERT computes the estimated inference accuracy $\hat{q}_{i,j}[\mathbf{T}_{\text{goal}}]$ by considering $t_{i,j}$ as a random variable that follows normal distribution with its mean and variance computed based on that of ξ. Here $q_{i,j}$ represents the inference accuracy when the DNN inference finishes before the deadline, and q_{fail} is the accuracy of a random guess:

$$\begin{aligned} \hat{q}_{i,j}[\mathbf{T}_{goal}] &= E(q_{i,j}[\mathbf{T}_{goal}] \mid t_{i,j}^{(n)}) \\ &= E(q_{i,j}[\mathbf{T}_{goal}] \mid \xi^{(n)} \cdot t_{i,j}^{\text{prof}}) \\ &= Pr_{i,j} \cdot q_{i,j} + (1 - Pr_{i,j}) \cdot q_{fail} \\ \xi^{(n)} &\sim \mathcal{N}(\mu^{(n)}, (\sigma^{(n)})^2) \end{aligned} \tag{7}$$

Energy. As discussed in Idea-3, ALERT predicts energy consumption by separately estimating energy during (1) DNN execution: estimated by multiplying the power limit by the estimated latency and (2) between inference inputs: estimated based on the recent history of inference idle power using the Kalman Filter in Eq. 8. $\phi^{(n)}$ is the predicted DNN-idle power ratio, $M^{(n)}$ is process variance, S is process noise, V is measurement noise, and $W^{(n)}$ is the Kalman Filter gain. ALERT initially sets $M^{(0)} = 0.01$, $S = 0.0001$, $V = 0.001$.

$$\begin{cases} W^{(n)} = \dfrac{M^{(n-1)} + S}{M^{(n-1)} + S + V} \\ M^{(n)} = (1 - W^{(n)})(M^{(n-1)} + S) \\ \phi^{(n)} = \phi^{(n-1)} + W^{(n)}(p_{\text{idle}}/p_{i,j}^{(n-1)} - \phi^{(n-1)}) \end{cases} \tag{8}$$

[3] A Kalman Filter is an optimal estimator that assumes a normal distribution and estimates a varying quantity based on multiple potentially noisy observations [55].

ALERT then predicts the energy by Eq. 9. Unlike Eq. 7 that uses probabilistic estimates, energy estimation is calculated without the notion of probability. The inference power is the same no matter the inference misses or meets the deadline, as ALERT sets power limits. Therefore it is safe to estimate the energy by its mean without considering the distribution of its possible latency. See our extended report [87] on estimating energy by its worst case latency percentile.

$$e_{i,j}^{(n)} = p_{i,j} \cdot \xi^{(n)} \cdot t_{i,j}^{\text{prof}} + \phi^{(n)} \cdot p_{i,j} \cdot (\mathbf{T}_{goal} - (\xi^{(n)} \cdot t_{i,j}^{\text{prof}})) \quad (9)$$

3.5 Integrating ALERT with Anytime DNNs

An anytime DNN is an inference model that outputs a series of increasingly accurate inference results—o_1, o_2, ... o_k, with o_t more reliable than o_{t-1}. A variety of recent works [35, 49, 52, 82, 86, 88] have proposed DNNs supporting anytime inference, covering a variety of problem domains. ALERT easily works with not only traditional DNNs but also Anytime DNNs. The only change is that q_{fail} in Eq. 3 no longer corresponds to a random guess. That is, when the inference could not generate its final result o_k by the deadline $\mathbf{T}_{\text{goal}}$, an earlier result o_x can be used with a much better accuracy than that of a random guess. The updated accuracy equation is below:

$$q_{.,j} = \begin{cases} q_k & \text{, if } t_{k,j} \leq \mathbf{t}_{\text{goal}} \\ q_{k-1} & \text{, if } t_{k-1,j} \leq \mathbf{t}_{\text{goal}} < t_{k,j} \\ & \cdots \\ q_{\text{fail}} & \text{, otherwise} \end{cases} \quad (10)$$

Existing anytime DNNs consider latency but not energy constraints—an anytime DNN will keep running until the latency deadline arrives and the last output will be delivered to the user. ALERT naturally improves Anytime DNN energy efficiency, stopping the inference sometimes before the deadline based on its estimation to meet not only latency and accuracy, but also energy requirements.

Furthermore, ALERT can work with a set of traditional DNNs and an Anytime DNN together to achieve the best combined result. The reason is that Anytime DNNs generally sacrifice accuracy for flexibility. When we feed a group of traditional DNNs and one Anytime DNN to construct the candidacy set $\mathbb{D}$, with Eq. 7, ALERT naturally selects the Anytime DNN when the environment is changing rapidly (because the expected accuracy of an anytime DNN will be higher given that variance), and the regular DNN, which has slightly higher accuracy with similar computation, when it is stable, getting the best of both worlds.

In our evaluation, we will use the nested design from [86], which provides a generic coverage of anytime DNNs.

3.6 Limitations and Discussions

Assumptions of the Kalman Filter. ALERT's prediction, particularly the Kalman Filter, relies on the feedback from recent input processing. Consequently, it requires at least one input to react to sudden changes. Additionally, the Kalman filter formulations assume that the underlying distributions are normal, which may not hold in practice. If the behavior is not Gaussian, the Kalman filter will produce bad estimations for the mean of ξ for some amount of time.

ALERT is specifically designed to handle data that is not drawn from a normal distribution, using the Kalman Filter's covariance estimation to measure system volatility and accounting for that in the accuracy/energy estimations. Consequently, after just 2–3 such bad predictions of means, the estimated variance will increase, which will then trigger ALERT to pick anytime DNN over traditional DNNs or pick a low-latency traditional DNN over high-latency ones, because the former has a higher expected accuracy under high variance. So—worst case—ALERT will choose a DNN with slightly less accuracy than what could have been used with the right model. Users can also compensate for extremely aberrant latency distributions by increasing the value of $Q^{(0)}$ in Eq. 5. Section 5.3 shows ALERT performs well even when the distribution is not normal.

Probabilistic guarantees. ALERT provides probabilistic, not hard, guarantees. As ALERT estimates not just average timing, but the distributions of possible timings, it can provide arbitrarily many nines of assurance that it will meet latency or accuracy goals but cannot provide 100% guarantee (see our extended report [87] on how to configure ALERT to provide guarantees with a specific probability). Providing 100% guarantees requires the worst case execution time (WCET), an upper bound on the highest possible latency. ALERT does not assume the availability of such information and hence cannot provide hard guarantees [6].

Safety guarantees. While ALERT does not explicitly model safety requirements, it can be configured to prioritize accuracy over other dimensions. When users particularly value safety (e.g., auto-driving), they could set a high accuracy requirement or even remove the energy constraints.

Concurrent inference jobs. ALERT is currently designed to support one inference job at a time. To support multiple concurrent inference jobs, future work needs to extend ALERT to coordinate across these concurrent jobs. We expect the main idea of ALERT, such as using a global slowdown factor to estimate system variation, to still apply.

Finally, how the inference behaves ultimately depends not only on ALERT, but also on the DNN models and system-resource setting options. As shown in Section 5, ALERT helps make the best use of supplied DNN models, but does not eliminate the difference between different DNN models.

4 Implementation

We implement ALERT for both CPUs and GPUs. On CPUs, ALERT adjusts power through Intel's RAPL interface [13], which allows software to set a hardware power limit. On

	Run-time environment setting		
Default	Inference task has no co-running process		
Memory	Co-locate with memory-hungry STREAM [60] (@CPU) Co-locate with Backprop from Rodinia-3.1 [8] (@GPU)		
Compute	Co-locate with Bodytrack from PARSEC-3.0 [5] (@CPU) Co-locate with the forward pass of Backprop [8] (@GPU)		
	Ranges of constraint setting		
Latency	0.4x–2x mean latency* of the largest Anytime DNN		
Accuracy	Whole range achievable by trad. and Anytime DNN		
Energy	Whole feasible power-cap ranges on the machine		
Task	Trad. DNN	Anytime [86]	Fixed deadline?
Image Classifi.	Sparse ResNet	Depth-Nest	Yes
Sentence Pred.	RNN	Width-Nest	No
Scheme ID	DNN selection		Power selection
Oracle	Dynamic optimal		Dynamic optimal
$\text{Oracle}_{\text{Static}}$	Static optimal		Static optimal
App-only	One Anytime DNN		System Default
Sys-only	Fastest traditional DNN		State-of-Art [37]
No-coord	Anytime DNN w/o coord. with Power		State-of-Art [37]
ALERT	ALERT default		ALERT default
$\text{ALERT}_{\text{Any}}$	ALERT w/o traditional DNNs		ALERT default
$\text{ALERT}_{\text{Trad}}$	ALERT w/o Anytime DNNs		ALERT default

Table 3: Settings and schemes under evaluation (* measured under default setting without resource contention)

GPUs, ALERT uses PyNVML to control frequency and builds a power-frequency lookup table. ALERT can also be applied to other approaches that translate power limits into settings for combinations of resources [33, 36, 72, 90].

In our experiments, ALERT considers a series of power settings within the feasible range with 2.5W interval on our test laptop and a 5W interval on our test CPU server and GPU platform, as the latter has a wider power range than the former. The number of power buckets is configurable.

ALERT incurs small overhead in both scheduler computation and switching from one DNN/power-setting to another, just 0.6–1.7% of an input inference time. We explicitly account for overhead by subtracting it from the user-specified goal (see step 2 in Section 3.2).

Users may set goals that are not achievable. If ALERT cannot meet all constraints, it prioritizes latency highest, then accuracy, then power. This hierarchy is configurable.

5 Experimental Evaluation

We apply ALERT to different inference tasks on both CPU and GPU with and without resource contention from co-located jobs. We set ALERT to (1) reduce energy while satisfying latency and accuracy requirements and (2) reduce error rates while satisfying latency and energy requirements. We compare ALERT with both oracle and state-of-the-art schemes and evaluate detailed design decisions.

5.1 Methodology

Experimental setup. We use the three platforms listed in Table 1: *CPU1*, *CPU2*, and *GPU*. On each, we run inference

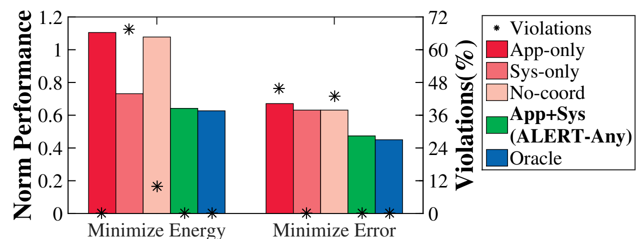


Figure 7: Average performance normalized to $\text{Oracle}_{\text{Static}}$. Violations% is %-of-constraint-settings under which a scheme incurs >10% violation of all inputs. (Smaller is better)

tasks[4], image classification and sentence prediction, under three different resource-contention scenarios:

- No contention: the inference task is the only job running, referred to as "Default";
- Memory dynamic: the inference task runs together with a memory-intensive job that repeatedly stops and restarts, representing dynamic memory resource contention, referred to as "Memory";
- Computation dynamic: the inference task runs together with a computation-intensive job that repeatedly stops and restarts, representing dynamic computation resource contention, referred to as "Compute".

Schemes in evaluation. We give ALERT three different DNN sets, traditional DNN models ($\text{ALERT}_{\text{Trad}}$), an Anytime DNN ($\text{ALERT}_{\text{Any}}$), and both (ALERT), and compare it with two oracle and three state-of-the-art schemes (Table 3).

The two $Oracle_*$ schemes have perfect predictions for every input under every DNN/power setting (i.e., impractical). Specifically, the "Oracle" allows DNN/power settings to change across inputs, representing the best possible results; the "$\text{Oracle}_{\text{Static}}$" has one fixed setting across inputs, representing the best results without dynamic adaptation.

The three state-of-the-art approaches include the following:

- "App-only" conducts adaptation only at the application level through an Anytime DNN [86];
- "Sys-only"adapts only at the system level following an existing resource-management system that minimizes energy under soft real-time constraints [62][5] and uses the fastest candidate DNN to avoid latency violations;
- "No-coord" uses *both* the Anytime DNN for application adaptation *and* the power-management scheme [62] to adapt power, but with these two working independently.

5.2 Overall Results

Table 4 shows the results for all schemes for different tasks on different platforms and environments. Each cell shows

[4] For GPU, we only run image classification task there, as the RNN-based sentence prediction task is better suited for CPU [91].

[5] Specifically, this adaptation uses a feedback scheduler that predicts inference latency based on Kalman Filter.

Plat.	DNN	Work.	ALERT	ALERT-Any	Sys-only	App-only	No-coord	Oracle	ALERT	ALERT-Any	Sys-only	App-only	No-coord	Oracle
			Energy in Minimizing Energy Task						Error Rate in Minimizing Error Task					
CPU1	Sparse Resnet	Idle	0.64	0.68	1.08^{19}	1.19	0.94^{1}	0.64	0.91	0.92	1.35	1.02^{3}	0.91^{3}	0.89
		Comp.	0.57	0.58	0.80^{19}	1.30	1.39^{1}	0.57	0.38	0.39	0.51	1.35^{24}	0.39^{6}	0.36
		Mem.	0.53	0.55	0.76^{19}	1.43	1.37^{2}	0.53	0.34	0.34	0.46	1.47^{28}	0.39^{2}	0.33
	RNN	Idle	0.61	0.65	1.01^{30}	1.34	0.95^{2}	0.61	0.87	0.87	0.87	0.87^{21}	0.87^{14}	0.86
		Comp.	0.60	0.57	0.93^{30}	1.21	1.26^{5}	0.60	0.42	0.44	0.50	0.46^{28}	0.46^{23}	0.42
		Mem.	0.54	0.56	0.95^{31}	1.45	1.24^{9}	0.54	0.45	0.45	0.50	0.57^{28}	0.54^{27}	0.44
CPU2	Sparse Resnet	Idle	0.93	0.88	0.96^{20}	0.99	1.18	0.91	0.68	0.68	0.97	0.79^{2}	0.71^{24}	0.66
		Comp.	0.59	0.57	0.60^{23}	1.00	1.01	0.58	0.58	0.57	0.85	0.74^{16}	0.71^{29}	0.55
		Mem.	0.38	0.37	0.39^{19}	0.65	0.63^{13}	0.38	0.24	0.82	0.32	0.33^{17}	0.75^{31}	0.21
	RNN	Idle	0.87	0.99	0.80^{34}	1.04	1.00^{6}	0.83	0.84	0.85	0.99	0.89^{14}	0.89^{1}	0.84
		Comp.	0.60	0.60	0.55^{34}	0.99	0.86^{7}	0.60	0.51	0.52	0.60	0.53^{21}	0.54^{17}	0.52
		Mem.	0.52	0.51	0.43^{33}	0.70	0.85^{14}	0.52	0.26	0.27	0.31	0.28^{21}	0.27^{17}	0.26
GPU	Sparse Resnet	Idle	0.97	0.99	0.92^{20}	1.36	1.37	0.92	0.90	0.92	1.22	1.09^{2}	1.74^{12}	0.86
		Comp.	0.96	0.97	0.94^{20}	1.66	1.77	0.89	0.32	0.34	1.28	1.21^{23}	2.50^{18}	0.30
		Mem.	0.97	1.01	0.91^{20}	1.39	1.43	0.91	0.89	0.92	1.22	1.11^{2}	1.81^{14}	0.86
Harmonic mean			0.64	0.64	0.73^{27}	1.11	1.08^{4}	0.62	0.46	0.47	0.63	0.67^{16}	0.63^{15}	0.45

Table 4: Average energy consumption and error rate normalized to $Oracle_{\text{Static}}$, smaller is better. (Each cell is averaged over 35–40 constraint settings; superscript: # of constraint settings violated for >10% inputs and hence excluded from energy average.)

the average energy or accuracy under 35–40 combinations of latency, accuracy, and energy constraints (the settings are detailed in Table 3), normalized to the $\text{Oracle}_{\text{Static}}$ result. Figure 7 compares these results, where lower bars represent better results and lower *s represent fewer constraint violations. ALERT and ALERT $_{\text{Any}}$ both work very well for all settings. They outperform state-of-the-art approaches, which have a significant number of constraint violations, as visualized by the many superscripts in Table 4 and the high * positions in Figure 7. ALERT outperforms $\text{Oracle}_{\text{Static}}$ because it adapts to dynamic variations. ALERT also comes very close to the theoretically optimal Oracle.

Comparing with Oracles. As shown in Table 4, ALERT achieves 93-99% of Oracle's energy and accuracy optimization while satisfying constraints. $\text{Oracle}_{\text{static}}$, the baseline in Table 4, represents the best one can achieve by selecting 1 DNN model and 1 power setting for all inputs. ALERT greatly out-performs $\text{Oracle}_{\text{static}}$, reducing its energy consumption by 3–48% while satisfying accuracy constraints (36% in harmonic mean) and reducing its error rate by 9-66% while satisfying energy constraints (54% in harmonic mean).

Figure 8 shows a detailed comparison for the energy minimization task. The figure shows the range of performance under all requirement settings (i.e., the whiskers). ALERT not only achieves similar mean energy reduction, its whole range of optimization behavior is also similar to Oracle. In comparison, $\text{Oracle}_{\text{Static}}$ not only has the worst mean but also the worst tail performance. Due to space constraints, we omit the figures for other settings, where similar trends hold.

ALERT has more advantage over $\text{Oracle}_{\text{static}}$ on CPUs than on GPUs. The CPUs have more empirical variance than the GPU, so they benefit more from dynamic adaptation. The GPU experiences significantly lower dynamic fluctuation so the static oracle makes good predictions.

ALERT satisfies the constraint in 99.9% of tests for image classification and 98.5% of those for sentence prediction. For the latter, due to the large input variability (NLP1 in Figure 4), some input sentences simply cannot complete by the deadline even with the fastest DNN. There the Oracle fails, too.

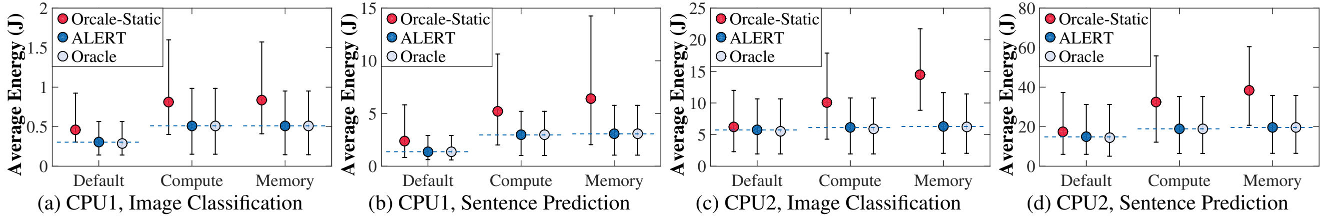
Note that, these Oracle schemes not only have perfect—and hence, impractical—prediction capability, but they also have no overhead. In contrast, ALERT is running on the same machines as the DNN workloads. *All results include ALERT's run-time latency and power overhead.*

Comparing with State-of-the-Art. For a fair comparison, we focus on $\text{ALERT}_{\text{Any}}$, as it uses exactly the same DNN candidate set as "Sys-only", "App-only", and "No-coord". Across all settings, $\text{ALERT}_{\text{Any}}$ outperforms the others.

The System-only solution suffers from not being able to choose different DNNs under different runtime scenarios. As a result, it performs much worse than $\text{ALERT}_{\text{Any}}$ in satisfying accuracy requirements or optimizing accuracy. For the former (left side of Table 4 and Figure 7), it creates accuracy violations in 68% of the settings as shown in Figure 7; for the latter (right side of Table 4 and Figure 7), although capable of satisfying energy constraints, it introduces 34% more error than $\text{ALERT}_{\text{Any}}$.

The Application-only solution that uses an Anytime DNN suffers from not being able to adjust to the energy requirements: it consumes 73% more energy in energy-minimizing tasks (left side of Table 4 and Figure 7) and introduces many energy-budget violations particularly under resource contention settings (right side of Table 4 and Fig. 7).

The no-coordination scheme is worse than both System- and Application-only. It violates constraints in both tasks with 69% more energy and 34% more error than $\text{ALERT}_{\text{Any}}$. Without coordination, the two levels can work at cross purposes; e.g., the application switches to a faster DNN to save energy while the system makes more power available.



(a) CPU1, Image Classification (b) CPU1, Sentence Prediction (c) CPU2, Image Classification (d) CPU2, Sentence Prediction

Figure 8: ALERT versus Oracle and $\text{Oracle}_{\text{Static}}$ on minimize energy task (Lower is better). (whisker: whole range; circle: mean)

Plat.	Work.	ALERT	Any	Trad	ALERT	Any	Trad
		Minimize Energy Task			Minimize Error Task		
CPU1	Idle	0.64	0.68	0.65^{1}	0.91	0.92	0.93
	Comp.	0.57	0.58	0.65^{6}	0.38	0.39	0.41
	Mem.	0.53	0.55	0.53^{3}	0.34	0.34	0.35
CPU2	Idle	0.93	0.88	0.95^{1}	0.68	0.68	0.69
	Comp.	0.59	0.57	0.60^{4}	0.58	0.57	0.59
	Mem.	0.38	0.37	0.40^{8}	0.23	0.24	0.32
GPU	Idle	0.97	0.99	0.95	0.90	0.92	0.89
	Comp.	0.97	1.01	0.96	0.89	0.92	0.89
	Mem.	0.96	0.97	0.95	0.32	0.34	0.32
Harmonic mean		0.66	0.66	0.67^{3}	0.47	0.48	0.50

Table 5: ALERT normalized average energy consumption and error rate to $\mathit{Oracle}_{\text{Static}}$ @ Sparse ResNet (Smaller is better)

5.3 Detailed Results and Sensitivity

Different DNN candidate sets. Table 5 compares the performance of ALERT working with an Anytime DNN (Any), a set of traditional DNN models (Trad), and both. At a high level, ALERT works well with all three DNN sets. Under close comparison, $\text{ALERT}_{\text{Trad}}$ violates more accuracy constraints than the others, particularly under resource contention on CPUs, because a traditional DNN has a much larger accuracy drop than an anytime DNN when missing a latency deadline. Consequently, when the system variation is large, $\text{ALERT}_{\text{Trad}}$ selects a faster DNN to meet latency and thus may not meet accuracy goals. Of course, $\text{ALERT}_{\text{Any}}$ is not always the best. As discussed in Section 3.5, Anytime DNNs sometimes have lower accuracy then a traditional DNN with similar execution time. This difference leads to the slightly better results for ALERT over $\text{ALERT}_{\text{Any}}$.

Figure 9 visualizes the different dynamic behavior of ALERT (blue curve) and $\text{ALERT}_{\text{Trad}}$ (orange curve) when the environment changes from Default to Memory-intensive and back. At the beginning, due to a loose latency constraint, ALERT and $\text{ALERT}_{\text{Trad}}$ both select the biggest traditional DNN, which provides the highest accuracy within the energy budget. When the memory contention suddenly starts, this DNN choice leads to a deadline miss and an energy-budget violation (as the idle period disappeared), which causes an accuracy dip. Fortunately, both quickly detect this problem and sense the high variability in the expected latency. ALERT switches to use an anytime DNN and a lower power cap. This switch is effective: although the environment is still unstable, the inference accuracy remains high, with slight ups and downs depending on which anytime output finished

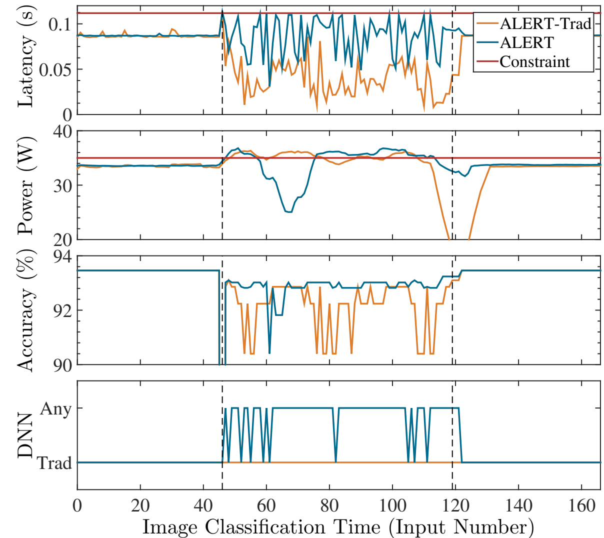


Figure 9: Minimize error rates w/ latency, energy constraints on CPU1. (Memory contention occurs from about input 46 to 119; Deadline: 1.25× mean latency of largest Anytime DNN in Default; power limit: 35W.)

before the deadline. Only able to choose from traditional DNNs, $\text{ALERT}_{\text{Trad}}$ conservatively switches to much simpler and hence lower-accuracy DNNs to avoid deadline misses. This switch does eliminate deadline misses under the highly dynamic environment, but many of the conservatively chosen DNNs finish before the deadline (see the Latency panel), wasting the opportunity to produce more accurate results and causing $\text{ALERT}_{\text{Trad}}$ to have a lower accuracy than ALERT. When the system quiesces, both schemes quickly shift back to the highest-accuracy, traditional DNN.

Overall, these results demonstrate how ALERT always makes use of the full potential of the DNN candidate set to optimize performance and satisfy constraints.

ALERT probabilistic design. A key feature of ALERT is its use of not just mean estimations, but also their variance. To evaluate the impact of this design, we compare ALERT to an alternative design ALERT*, which only uses the estimated mean to select configurations.

Figure 10 shows the performance of ALERT and ALERT* in the minimize error task for sentence prediction. Here, ALERT (blue circles) always performs better than ALERT*. Its advantage is the biggest when the DNN candidates include both traditional and Anytime DNNs (i.e., the "Standard"

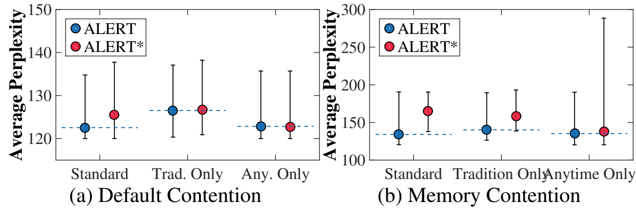


Figure 10: Minimize error for sentence prediction@ CPU1 (Lower is better). (whisker: whole range; circle: mean)

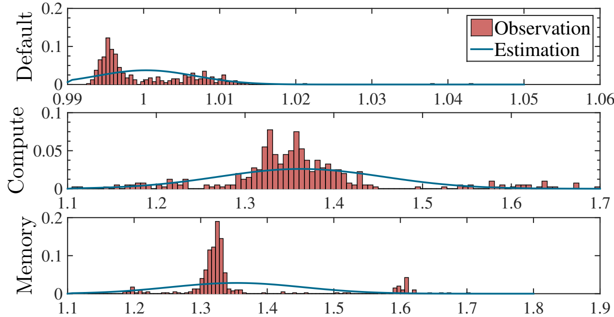


Figure 11: Distribution of ξ for image class. on CPU1.

in Figure 10). The reason is that traditional DNNs and Anytime DNN have different accuracy/latency curves, Eq. 3 for the former and Eq. 10 for the latter. ALERT* is much worse in distinguishing these two by simply using the mean of estimated latency to predict accuracy. ALERT also clearly outperforms ALERT* under memory contention with traditional DNN candidates, as ALERT's estimation better captures dynamic system variation. Overall, these results show ALERT's probabilistic design is effective.

Sensitivity to latency distribution. ALERT assumes a Gaussian distribution, but is designed to work for other distributions (see Section 3.6). As shown in Figure 11, the observed ξs (red bars) are indeed not a perfect fit for Gaussian distribution (blue lines), which confirms ALERT's robustness.

6 Related work

Past resource management systems have used machine learning [4, 51, 68, 69, 79] or control theory [32, 37, 44, 45, 62, 74, 93] to make dynamic decisions and adapt to changing environments or application needs. Some also use Kalman filter because it has optimal error properties [37, 44, 45, 62]. There are two major differences between them and ALERT: 1) prior approaches use the Kalman filter to estimate physical quantities such as CPU utilization [45] or job latency [37], while ALERT estimates a *virtual* quantity that is then used to update a large number of latency estimates. 2) while variance is naturally computed as part of the filter, ALERT actually uses it, in addition to the mean, to help produce estimates that better account for environment variability.

Past work designed resource managers explicitly to coordinate approximate applications with system resource usage [21, 31, 32, 46]. Although related, they manage applications *separately* from system resources, which is fundamentally different from ALERT's holistic design. When an environmental change occurs, prior approaches first adjust the application and then the system serially (or vice versa) so that the change's effects on each can be established independently [31, 32]. That is, coordination is established by forcing one level to lag behind the other. In practice this design forces each level to keep its own independent model and delays response to environmental changes. In contrast, ALERT's global slowdown factor allows it to easily model and update prediction about all application and system configurations simultaneously, leading to very fast response times, like the single input delay demonstrated in Figure 9.

Much work accelerates DNNs through hardware [3, 10–12, 19, 23, 24, 27, 30, 38, 43, 54, 58, 66, 73, 75, 83], compiler [9, 65], system [28, 53], or design support [25, 25, 26, 39, 42, 77, 81, 85]. They essentially shift and extend the tradeoff space, but do not provide policies for meeting user needs or for navigating tradeoffs dynamically, and hence are orthogonal to ALERT.

Some research supports hard real-time guarantees for DNNs [92], providing 100% timing guarantees while assuming that the DNN model gives the desired accuracy, the environment is completely predictable, and energy consumption is not a concern. ALERT provides slightly weaker timing guarantees, but manages accuracy and power goals. ALERT also provides more flexibility to adapt to unpredictable environments. Hard real-time systems would fail in the co-located scenario unless they explicitly account for all possible co-located applications at design time.

7 Conclusion

This paper demonstrates the challenges behind the important problem of ensuring timely, accurate, and energy efficient neural network inference with dynamic input, contention, and requirement variation. ALERT achieves these goals through dynamic and coordinated DNN model selection and power management based on feedback control. We evaluate ALERT with a variety of workloads and DNN models and achieve high performance and energy efficiency.

Acknowledgement

We thank the reviewers for their helpful feedback and Ken Birman for shepherding this paper. This research is supported by NSF (grants CNS-1956180, CNS-1764039, CNS-1764039, CNS-1514256, CNS-1823032, CCF-1439156), ARO (grant W911NF1920321), DOE (grant DESC0014195 0003), DARPA (grant FA8750-16-2-0004) and the CERES Center for Unstoppable Computing. Additional support comes from the DARPA BRASS program and a DOE Early Career award.

References

[1] Baidu AI. Apollo open vehicle certificate platform. Online document, http://apollo.auto, 2018.

[2] S. Akhlaghi, N. Zhou, and Z. Huang. Adaptive adjustment of noise covariance in kalman filter for dynamic state estimation. In *IEEE Power Energy Society General Meeting*, 2017.

[3] Jorge Albericio, Patrick Judd, Tayler Hetherington, Tor Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Cnvlutin: Ineffectual-neuron-free deep neural network computing. In *ISCA*, pages 1–13, 2016.

[4] Jason Ansel, Maciej Pacula, Yee Lok Wong, Cy Chan, Marek Olszewski, Una-May O'Reilly, and Saman Amarasinghe. Siblingrivalry: online autotuning through local competitions. In *CASES*, 2012.

[5] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *PACT*, October 2008.

[6] Giorgio C Buttazzo, Giuseppe Lipari, Luca Abeni, and Marco Caccamo. *Soft Real-Time Systems: Predictability vs. Efficiency: Predictability vs. Efficiency*. Springer, 2006.

[7] Aaron Carroll and Gernot Heiser. Mobile multicores: Use them or waste them. In *HotPower*, 2013.

[8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, 2009.

[9] Tianqi Chen, Thierry Moreau, Ziheng Jiang, Lianmin Zheng, Eddie Yan, Haichen Shen, Meghan Cowan, Leyuan Wang, Yuwei Hu, Luis Ceze, et al. Tvm: An automated end-to-end optimizing compiler for deep learning. In *OSDI*, pages 578–594, 2018.

[10] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. *SIGPLAN Not.*, pages 269–284, 2014.

[11] Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *JSSC*, 2016.

[12] Yunji Chen, Tao Luo, Shaoli Liu, Shijin Zhang, Liqiang He, Jia Wang, Ling Li, Tianshi Chen, Zhiwei Xu, Ninghui Sun, et al. Dadiannao: A machine-learning supercomputer. In *MICRO 47*, pages 609–622, 2014.

[13] H. David, E. Gorbatov, U. R. Hanebutte, R. Khanna, and C. Le. Rapl: Memory power estimation and capping. In *ISLPED*, 2010.

[14] Christina Delimitrou and Christos Kozyrakis. Paragon: Qos-aware scheduling for heterogeneous datacenters. In *ASPLOS*, 2013.

[15] Christina Delimitrou and Christos Kozyrakis. Quasar: resource-efficient and qos-aware cluster management. In *ASPLOS*, 2014.

[16] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *TPAMI*, 2011.

[19] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. Shidiannao: Shifting vision processing closer to the sensor. In *ISCA*, pages 92–104, 2015.

[20] Biyi Fang, Xiao Zeng, and Mi Zhang. Nestdnn: Resource-aware multi-tenant on-device deep learning for continuous mobile vision. In *Mobicom*, 2018.

[21] Anne Farrell and Henry Hoffmann. MEANTIME: achieving both minimal energy and timeliness with approximate computing. In *USENIX ATC*, 2016.

[22] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry P Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, page 7, 2017.

[23] Mingyu Gao, Christina Delimitrou, Dimin Niu, Krishna T Malladi, Hongzhong Zheng, Bob Brennan, and Christos Kozyrakis. Draf: a low-power dram-based reconfigurable acceleration fabric. *ISCA*, pages 506–518, 2016.

[24] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *ISCA*, pages 243–254, 2016.

[25] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[26] Soheil Hashemi, Nicholas Anthony, Hokchhay Tann, R Iris Bahar, and Sherief Reda. Understanding the impact of precision quantization on the accuracy and energy of neural networks. In *DATE*, pages 1474–1479, 2017.

[27] Johann Hauswald, Yiping Kang, Michael A Laurenzano, Quan Chen, Cheng Li, Trevor Mudge, Ronald G Dreslinski, Jason Mars, and Lingjia Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *ISCA*, pages 27–40, 2015.

[28] Johann Hauswald, Michael A Laurenzano, Yunqi Zhang, Cheng Li, Austin Rovinski, Arjun Khurana, Ronald G Dreslinski, Trevor Mudge, Vinicius Petrucci, Lingjia Tang, et al. Sirius: An open end-to-end voice and vision personal assistant and its implications for future warehouse scale computers. In *ASPLOS*, pages 223–238, 2015.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[30] Parker Hill, Animesh Jain, Mason Hill, Babak Zamirai, Chang-Hong Hsu, Michael A Laurenzano, Scott Mahlke, Lingjia Tang, and Jason Mars. Deftnn: Addressing bottlenecks for dnn execution on gpus via synapse vector elimination and near-compute data fission. In *MICRO*, pages 786–799, 2017.

[31] Henry Hoffmann. Coadapt: Predictable behavior for accuracy-aware applications running on power-aware systems. In *ECRTS*, pages 223–232, 2014.

[32] Henry Hoffmann. Jouleguard: energy guarantees for approximate applications. In *SOSP*, 2015.

[33] Henry Hoffmann and Martina Maggio. PCP: A generalized approach to optimizing performance under power constraints through resource management. In *ICAC*, pages 241–247, 2014.

[34] Hanzhang Hu, Debadeepta Dey, Martial Hebert, and J Andrew Bagnell. Learning anytime predictions in neural networks via adaptive loss balancing. In *AAAI*, 2019.

[35] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. In *CoRR*, 2017.

[36] C. Imes and H. Hoffmann. Bard: A unified framework for managing soft timing and power constraints. In *SAMOS*, pages 31–38, 2016.

[37] C. Imes, D. H. K. Kim, M. Maggio, and H. Hoffmann. Poet: a portable approach to minimizing energy under soft real-time constraints. In *RTAS*, pages 75–86, April 2015.

[38] Animesh Jain, Michael A Laurenzano, Gilles A Pokam, Jason Mars, and Lingjia Tang. Architectural support for convolutional neural networks on modern cpus. In *PACT*, 2018.

[39] Shubham Jain, Swagath Venkataramani, Vijayalakshmi Srinivasan, Jungwook Choi, Pierce Chuang, and Leland Chang. Compensated-dnn: energy efficient low-precision deep neural networks by compensating quantization errors. In *DAC*, pages 1–6, 2018.

[40] Junchen Jiang, Ganesh Ananthanarayanan, Peter Bodik, Siddhartha Sen, and Ion Stoica. Chameleon: scalable adaptation of video analytics. In *ACM SIGCOMM*, pages 253–266, 2018.

[41] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[42] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, Natalie Enright Jerger, and Andreas Moshovos. Proteus: Exploiting numerical precision variability in deep neural networks. In *ICS*, page 23, 2016.

[43] Patrick Judd, Jorge Albericio, Tayler Hetherington, Tor M Aamodt, and Andreas Moshovos. Stripes: Bit-serial deep neural network computing. In *MICRO*, pages 1–12, 2016.

[44] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters. In *ICAC*, 2009.

[45] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Adaptive resource provisioning for virtualized servers using kalman filters. *TAAS*, 2014.

[46] Aman Kansal, Scott Saponas, AJ Brush, Kathryn S McKinley, Todd Mytkowicz, and Ryder Ziola. The latency, accuracy, and battery (lab) abstraction: programmer productivity and energy efficiency for continuous mobile context sensing. In *OOPSLA*, 2013.

[47] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *ICCPS*, pages 287–296, 2018.

[48] D. H. K. Kim, C. Imes, and H. Hoffmann. Racing and pacing to idle: Theoretical and empirical analysis of energy optimization heuristics. In *ICCPS*, 2015.

[49] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

[50] Etienne Le Sueur and Gernot Heiser. Slow down or sleep, that is the question. In *USENIX ATC*, June 2011.

[51] Benjamin C Lee and David Brooks. Efficiency trends and limits from comprehensive microarchitectural adaptivity. *ASPLOS*, 2008.

[52] Hankook Lee and Jinwoo Shin. Anytime neural prediction via slicing networks vertically. *arXiv preprint arXiv:1807.02609*, 2018.

[53] Shih-Chieh Lin, Yunqi Zhang, Chang-Hong Hsu, Matt Skach, Md E Haque, Lingjia Tang, and Jason Mars. The architectural implications of autonomous driving: Constraints and acceleration. In *ASPLOS*, pages 751–766, 2018.

[54] Daofu Liu, Tianshi Chen, Shaoli Liu, Jinhong Zhou, Shengyuan Zhou, Olivier Teman, Xiaobing Feng, Xuehai Zhou, and Yunji Chen. Pudiannao: A polyvalent machine learning accelerator. In *ISCA*, pages 369–381, 2015.

[55] Jun S Liu and Rong Chen. Sequential monte carlo methods for dynamic systems. *Journal of the American statistical association*, 1998.

[56] ATLAS LS. What is simultaneous/conference interpretation? Online document, https://atlasls.com/what-is-simultaneousconference-interpretation/, 2010.

[57] Martina Maggio, Alessandro Vittorio Papadopoulos, Antonio Filieri, and Henry Hoffmann. Automated control of multiple software goals using multiple actuators. In *FSE*, 2017.

[58] Divya Mahajan, Jongse Park, Emmanuel Amaro, Hardik Sharma, Amir Yazdanbakhsh, Joon Kyung Kim, and Hadi Esmaeilzadeh. Tabla: A unified template-based framework for accelerating statistical machine learning. In *HPCA*, pages 14–26. IEEE, 2016.

[59] Mitchell P. Marcus, Beatrice Santorini, Mary Ann Marcinkiewicz, and Ann Taylor. Treebank-3 - linguistic data consortium. Online document, https://catalog.ldc.upenn.edu/LDC99T42, 1999.

[60] John D McCalpin. Memory bandwidth and machine balance in current high performance computers. *TCCA*, 1995.

[61] Mason McGill and Pietro Perona. Deciding how to decide: Dynamic routing in artificial neural networks. *arXiv preprint arXiv:1703.06217*, 2017.

[62] Nikita Mishra, Connor Imes, John D. Lafferty, and Henry Hoffmann. CALOREE: learning control for predictable latency and low energy. In *ASPLOS*, 2018.

[63] Nikita Mishra, Huazhe Zhang, John D. Lafferty, and Henry Hoffmann. A probabilistic graphical model-based approach for minimizing energy under performance constraints. *ASPLOS*, 2015.

[64] Jakob Nielsen. *Usability engineering*. Elsevier, 1994.

[65] NVIDIA. Nvidia tensorrt: Programmable inference accelerator. Online document, https://developer.nvidia.com/tensorrt, 2018.

[66] Kalin Ovtcharov, Olatunji Ruwase, Joo-Young Kim, Jeremy Fowers, Karin Strauss, and Eric S Chung. Accelerating deep convolutional neural networks using specialized hardware. *Microsoft Research Whitepaper*, 2015.

[67] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *SOSP*, 2017.

[68] Paula Petrica, Adam M Izraelevitz, David H Albonesi, and Christine A Shoemaker. Flicker: A dynamically adaptive architecture for power limited multicore systems. In *ISCA*, 2013.

[69] Dmitry Ponomarev, Gurhan Kucuk, and Kanad Ghose. Reducing power requirements of instruction scheduling through dynamic allocation of multiple datapath resources. In *MICRO*, 2001.

[70] Amir M. Rahmani, Bryan Donyanavard, Tiago Mück, Kasra Moazzemi, Axel Jantsch, Onur Mutlu, and Nikil D. Dutt. SPECTR: formal supervisory control and coordination for many-core systems resource management. In *ASPLOS*, pages 169–183, 2018.

[71] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250*, 2016.

[72] S. Reda, R. Cochran, and A. K. Coskun. Adaptive power capping for servers with multithreaded workloads. *MICRO*, 2012.

[73] Minsoo Rhu, Natalia Gimelshein, Jason Clemons, Arslan Zulfiqar, and Stephen W Keckler. vdnn: Virtualized deep neural networks for scalable, memory-efficient neural network design. In *MICRO*, page 18, 2016.

[74] Muhammad Husni Santriaji and Henry Hoffmann. Grape: Minimizing energy for gpu applications with performance requirements. In *MICRO*, 2016.

[75] Hardik Sharma, Jongse Park, Divya Mahajan, Emmanuel Amaro, Joon Kyung Kim, Chenkai Shao, Asit Mishra, and Hadi Esmaeilzadeh. From high-level deep neural models to fpgas. In *MICRO*, page 17, 2016.

[76] N Silberman and Guadarrama. S. Tensorflow-slim image classification model library. Online document, https://github.com/tensorflow/models/tree/master/research/slim, 2016.

[77] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. Dps: dynamic precision scaling for stochastic computing-based deep neural networks. In *DAC*, page 13, 2018.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[79] Srinath Sridharan, Gagan Gupta, and Gurindar S Sohi. Holistic run-time parallelism management for time and energy efficiency. In *ICS*, 2013.

[80] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *ASE*, 2018.

[81] Hokchhay Tann, Soheil Hashemi, R Iris Bahar, and Sherief Reda. Hardware-software codesign of accurate, multiplier-free deep neural networks. In *DAC*, 2017.

[82] Surat Teerapittayanon, Bradley McDanel, and H.T. Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *CVPR*, 2016.

[83] Vincent Vanhoucke, Andrew Senior, and Mark Z Mao. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, page 4, 2011.

[84] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *ECCV*, 2018.

[85] Swagath Venkataramani, Ashish Ranjan, Kaushik Roy, and Anand Raghunathan. Axnn: energy-efficient neuromorphic systems using approximate computing. In *ISLPED*, 2014.

[86] Chengcheng Wan, Henry Hoffmann, Shan Lu, and Michael Maire. Orthogonalized SGD and nested architectures for anytime neural networks. In *ICML 2020, to appear*.

[87] Chengcheng Wan, Muhammad Santriaji, Eri Rogers, Henry Hoffmann, Michael Maire, and Shan Lu. Alert: Accurate learning for energy and timeliness. *arXiv preprint arXiv:1911.00119*, 2020.

[88] Yan Wang, Zihang Lai, Gao Huang, Brian H Wang, Laurens van der Maaten, Mark Campbell, and Kilian Q Weinberger. Anytime stereo image depth estimation on mobile devices. *arXiv preprint arXiv:1810.11408*, 2018.

[89] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *CVPR*, pages 8817–8826, 2018.

[90] Huazhe Zhang and Henry Hoffmann. Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. In *ASPLOS*, 2016.

[91] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. Deepcpu: Serving rnn-based deep learning models 10x faster. In *ATC*, pages 951–965, 2018.

[92] H. Zhou, S. Bateni, and C. Liu. S3dnn: Supervised streaming and scheduling for gpu-accelerated real-time DNN workloads. In *RTAS*, 2018.

[93] Yanqi Zhou, Henry Hoffmann, and David Wentzlaff. Cash: Supporting iaas customers with a sub-core configurable architecture. In *ISCA*, 2016.