Precious: Resource-Demand Estimation for Embedded Neural Network Accelerators

Stefan Reif, Benedict Herzog, Judith Hemp, Timo Hönig, and Wolfgang Schröder-Preikschat
Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU)
{reif,benedict.herzog,hemp,thoenig,wosch}@cs.fau.de

ABSTRACT

The recent advances of hardware-based accelerators for machine learning—in particular neural networks—attracted the attention of embedded-systems designers and engineers. Since embedded systems usually operate with strict resource constraints, knowledge about the resource demand (i.e., time and power) for executing machine-learning workloads is key. This paper presents Precious, an approach, as well as a practical implementation, of a system that estimates execution time and power draw of convolutional and fully-connected neural networks that execute on a commercially-available off-the-shelf embedded accelerator hardware for neural networks (i.e., Google Coral Edge TPU).

ACM Reference Format:
Stefan Reif, Benedict Herzog, Judith Hemp, Timo Hönig, and Wolfgang Schröder-Preikschat. 2020. Precious: Resource-Demand Estimation for Embedded Neural Network Accelerators. In *First International Workshop on Benchmarking Machine Learning Workloads on Emerging Hardware, March 4, 2020, Austin, TX*. ACM, New York, NY, USA, 9 pages.

1 INTRODUCTION

Hardware accelerators for machine learning (e.g., Intel Neural Compute Stick [9], Google Coral [22]) are increasingly common on a wide variety of different platforms. The hardware accelerators satisfy the growing demand and interest for machine-learning approaches [1, 21]. The corresponding machine-learning workloads implement application scenarios (e.g., the use of image classification and speech recognition) that execute more efficiently using the accelerator-specific integrated circuits.

However, as the platforms for such tasks (i.e., tablet computers, smart phones, watches, and other wearable devices) often operate on strictly limited resources [8, 13–15] machine-learning workloads [2, 3] must be analysed with regards to their power and energy demand as they impact the battery life of the host systems. The adaptation of machine-learning workloads for new application scenarios increases the pressure at system level (i.e., system software [23], operating systems [24]) to integrate machine-learning accelerators efficiently. When machine-learning workloads execute, it is important to consider non-functional system properties such as the battery life of the host system. This is especially relevant as the use of hardware-based accelerators [9, 22] becomes a state-of-the-art technique in computer science and finds broad adoption.

Until now, estimating the energy demand of machine-learning workloads is a field of research that is explored very little. Thus, it remains an unresolved challenge to predict the total energy demand for executing a machine-learning workload—for example, the energy demand that is required to execute a specific neural network with a given input.

A straight-forward way to determine the power and energy demand for the execution of machine-learning workloads is to measure it, for example, with a power meter. However, such power measurements depend on the availability of the final hardware (i.e., accelerator modules), measurement infrastructure (i.e., power meters) that can be costly, and hyper-parameters that are fine-tuned during an expensive—with respect to time and energy—training process. Therefore, it is most desirable to estimate the energy demand *before* all components of the final system are available. In particular, system developers must define neural-network parameters prior to the resource-intensive training process. However, these parameters influence non-functional system properties, such as latencies and energy demand. Hence, a resource-demand model enables developers to assess the influence of parameter choices and revise bad choices before spending resources on the training process, needlessly.

This paper tackles the challenge of analysing and estimating the energy demand of machine-learning workloads. We analyse the power and energy demand of a hardware accelerator for machine learning (i.e., Google Coral [22]). Based on the conducted energy measurements at the hardware level, we present and discuss the concept and implementation of Precious, an approach for the estimation of the energy demand for machine-learning workloads.

The contributions of the paper are threefold. First, we present and discuss an in-depth analysis for the energy demand of machine-learning workloads. Second, we propose Precious, an approach for the estimation of the energy demand of machine-learning workloads with models based on linear and random forest regressors. Third, we evaluate our current implementation of Precious and discuss the differences between linear and random forest models. In order to provide the necessary base data for our work, we conduct measurements for a large series of different machine-learning workloads (i.e., deep artificial neural networks with different network structure and input) and provide an in-depth discussion of the resulting energy demand for the various different workloads. We further present Precious, an approach for modelling and estimating the energy demand of deep artificial neural networks at a fine-granular level. Our experiments use the Google Coral Edge accelerator [22], a commercially available off-the-shelf (COTS) neural network acceleration hardware module for which we conduct energy measurements at the hardware level.

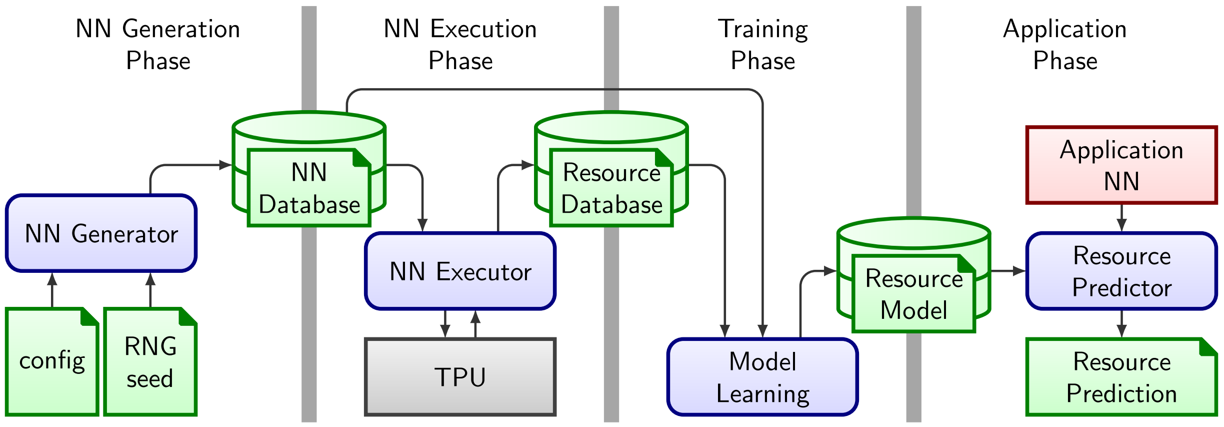

Figure 1: Precious comprises four phases for the estimation of execution time and power draw of neural networks.

The paper is structured as follows. Section 2 presents the Precious approach to estimate the resource demand of neural networks and our practical implementation. Section 3 evaluates the system empirically. Section 4 and Section 5 discuss related work and future work, respectively. Section 6 concludes this paper.

2 DESIGN AND IMPLEMENTATION

In this section we present the Precious approach for the resource estimation of neural networks (NN). We outline design considerations of Precious and discuss its current implementation which uses a Google Coral Edge Tensor Processing Unit (TPU), a COTS neural network accelerator for embedded systems that connects to its host system via USB [22]. Precious organises the process of estimating the resource demand of neural networks in four distinct phases (see Figure 1). First, Precious generates randomised neural networks. Second, it evaluates their resource demands. Third, a machine-learning approach builds a model that maps network properties (e.g., number of layers) to its resource demand. In the fourth phase, applications can *utilise* this model to estimate the resource demand of their networks.

2.1 Neural Network Generation

In the first phase, Precious uses the machine-learning framework TensorFlow 2.0 with Keras [21] and Python 3.6 to generate randomised neural networks. It currently supports two different network types—convolutional (“conv2d”) and fully connected (“dense”) networks [17]. As of now, the implementation supports “relu” as activation function, and imposes restrictions on the dimensions of convolutional networks. For each layer, the input dimension is the same as the output dimension. Each generated network consists of homogeneous layers of the same type and dimension (for convolutional layers this implies the use of the “same” padding). Convolutional layers always have a square-dimension input with a depth of 3, and 3 filters with the dimension $(3, 3)$. Our system varies the number of layers (between 2 and 250) and the dimensions (between 100 and 1024), randomly. Furthermore, all network-internal parameters are initialised to random values. We do not apply a training algorithm, as we use Precious to examine the resource demand of different network types and configurations and not the resource demand for a specifically trained network.

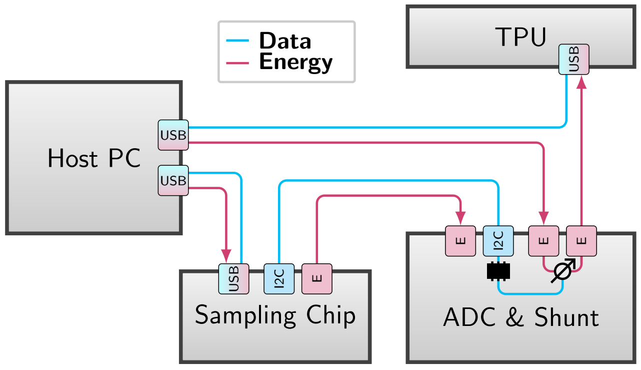

Figure 2: The energy-demand measurement setup intercepts the power supply of the accelerator at its USB connection.

To execute a neural network on the accelerator, it must be converted to a suitable TensorFlow Lite (TFLite) model. This means, for example, that all parameters must be in an 8-bit fixed-point number format. The conversion to this format is either achieved during training (i.e., “quantisation-aware training”) or afterwards (i.e., “post-training quantisation”). Since Precious only pretends that the generated neural networks are trained, it uses the latter technique to translate the randomised parameters to 8-bit values. Furthermore, this technique also allows to convert existing neural networks that are already trained and execute them on the accelerator. This network translation process is executed on the host system using the Edge TPU compiler [18]. The compiler also reveals hardware-related network features, such as its memory demand on the accelerator.

2.2 Neural Network Execution

In the second phase, Precious measures the energy demand of generated neural networks. Figure 2 visualises the structure of our

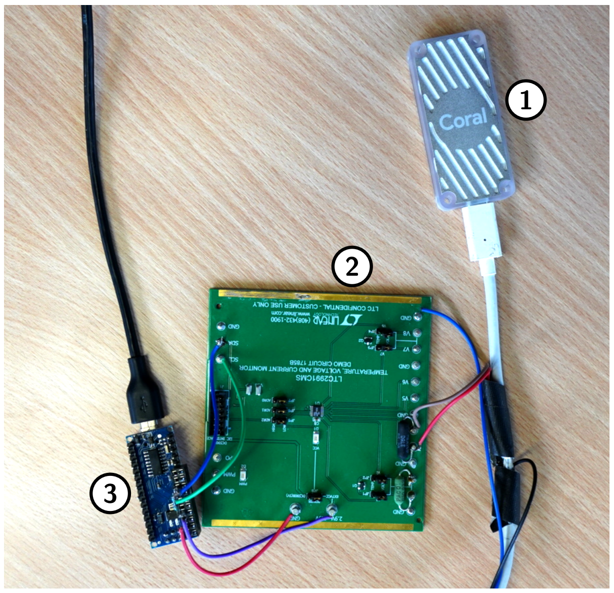

Figure 3: Picture of the energy-demand measurement setup with ① the TPU, ② ADC and shunt, and ③ the sampling chip.

energy measurement setup, and Figure 3 shows a picture of the actual hardware setup. The host system submits neural networks to the Coral Edge accelerator [22] for execution[1]. Thereby, the input data for the execution is randomised and no batching is applied. Precious intercepts the power supply of the accelerator at its USB connection. It uses a shunt resistor together with a LTC2991 voltage and current monitoring sensor [5] which has a 14-bit ADC to measure the power draw. A microcontroller ("sampling chip") polls the power measurement values, aggregates them, and eventually communicates them to the host system.

This setup allows Precious to create detailed power traces, as shown in Figure 4. According to our measurements, the accelerator's idle power draw is between 280 mW and 300 mW. Even before the actual inference begins, we observe significant fluctuations in power demand that go hand in hand with the creation of the TFLite interpreter object. This software object is necessary for the execution of the model [19] and switches the accelerator to a state of increased power demand. During the first 2 seconds of the measurement we observe power fluctuations *ahead* of the actual inference execution (at around 3 seconds into the measurement) as visualised in Figure 4. This means that there exists a pre-processing stage before the first execution of an inference.

The TPU executes inferences only during the middle part of the curve (in Figure 4, beginning at around 3 seconds until ca. 6.5 seconds). During the first inference, the neural network gets loaded onto the TPU, so it lasts longer than all other inferences [20], and therefore, its measurement results are excluded from further processing. The subsequent inferences yield measurement results from which Precious derives its power and time models. After the last inference, the interpreter object is destroyed and the accelerator returns to the idle state after some time. This post-processing stage still has a slightly higher power draw compared to the TPU's initial idle state. We assume that the pre- and post-processing stage correspond to different USB transmission and power modes of the Cortex-M0+ microprocessor that operates on the Google Coral TPU.

In Figure 5, the measured values of several inferences of the same convolutional networks were superimposed to visualise the power course of an inference. The superimposed visualisation allows a detailed analysis of the power draw over time, even though a single inference executes within a few milliseconds. This is due to the deterministic execution behaviour of the TPU [11]. The measurements show clear differences between dense and convolutional networks. Convolutional networks have clearly repeating "hills", whereas dense networks start with a striking spike to end in slight sine curves, as shown in Figure 6. For both network types, the observed behaviour is repeatable.

For execution on the accelerator, a delegate of the TFLite interpreter of the manufacturers is required, which is implemented by the program library "libedgetpu.so". For the embedded TPU of our implementation [22], this library is available in different two versions, a standard variant and a second variant which operates with the maximum operating frequency [20]. Depending on the installed version, the power demand and execution time of the TPU differs. We use the former (i.e., slower) variant for our implementation of Precious because the latter suffers from thermal problems. For each network, inferences are repeatedly carried out for at least one minute, while measuring execution time and power demand. Then, Precious computes the average power draw and the average execution time for each generated neural network.

2.3 Model Training

To estimate the power draw and execution time for the inference of neural networks, Precious uses linear [26] and random forest regressors [27] from scikit-learn. Both are well-established stochastic modelling techniques.

The training data set consists of 1016 dense neural network and 415 convolutional neural networks. We take 80 % of the generated neural networks for training, the other 20 % constitute the evaluation set used in Section 3. For each neural network in the training set, we collect statically known features as follows and measure their power draw and execution times.

The following properties were provided as input to the learned model: number of layers, number of network parameters, memory demand (host and on-chip, separately), and the number of multiply-accumulate (MAC) operations. Each parameter is easily determined before executing the neural network on the embedded TPU. The number of layers is an inherent property of the underlying network, whereas the number of network parameters results from the number of neurons and filter size for convolutional networks. With this information, the number of MAC operations is calculated. The outputs of the models are the measured power draw and execution time. Precious uses scikit-learn to fit linear regressors and random forest regressors to these data sets, one model for convolutional networks and another model for dense networks, respectively. The optimal hyper parameters for the random forest models

[1]Throughout this paper, a "neural network execution" is equivalent to computing an inference with one input data set.

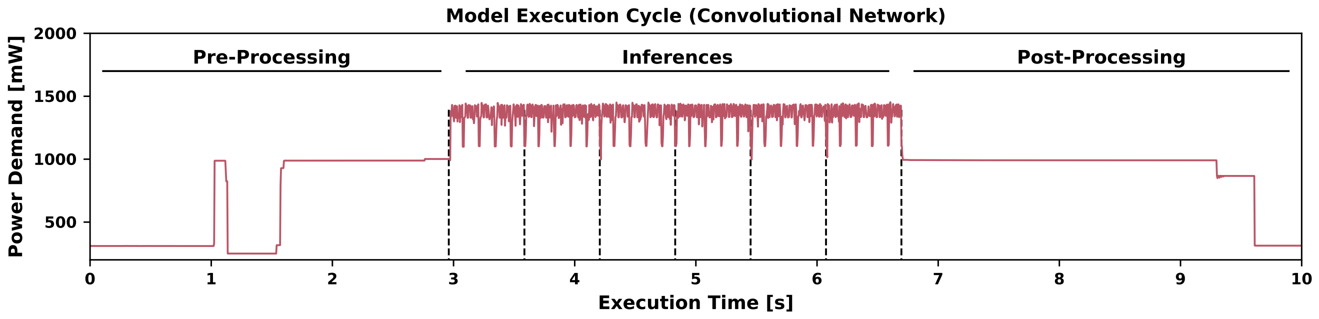

Figure 4: Power curve of full model execution, including the pre-processing stage, six inference runs (dashed lines), and the post-processing stage. The executed convolutional network consists of 101 layers and a dimension of (1019,1019,3).

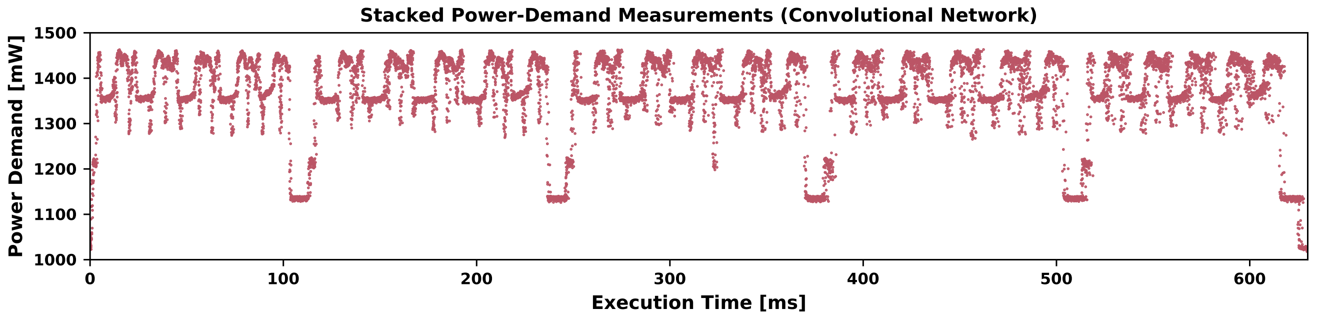

Figure 5: Power demand during inference execution for the convolutional network as shown in Figure 4. The graph was plotted with stacked power measurements of 113 inference iterations.

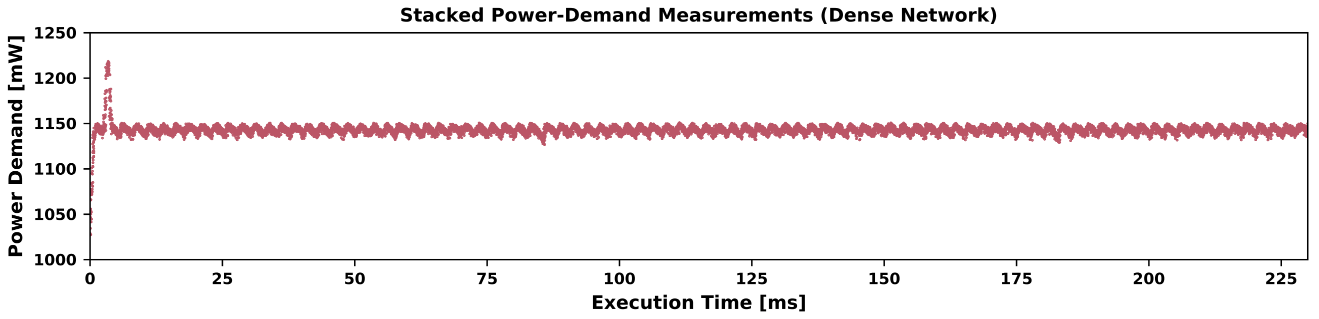

Figure 6: Power demand during inference execution for a dense neural network consisting of 101 layers and a dimension of 1071. The graph was plotted with stacked power measurements of 32 inference iterations.

were determined using a randomised search [28] with the training data set.

2.4 Model Application

In the fourth and final phase, applications *apply* the trained models to estimate the resource demand of their neural networks. The intended use-case is that phases 1 to 3 are executed once, for example by the hardware vendor, and the resulting model is distributed. In contrast to cycle-accurate simulators, such measurement-based resource-demand models can be published without the risk of leaking intellectual property. Using these models, developers can determine the resource demand of different network architectures—without power measurement infrastructure and before training—and restrict the training process to models that satisfy all resource constraints. The following section evaluates the quality of resource-demand estimation by Precious.

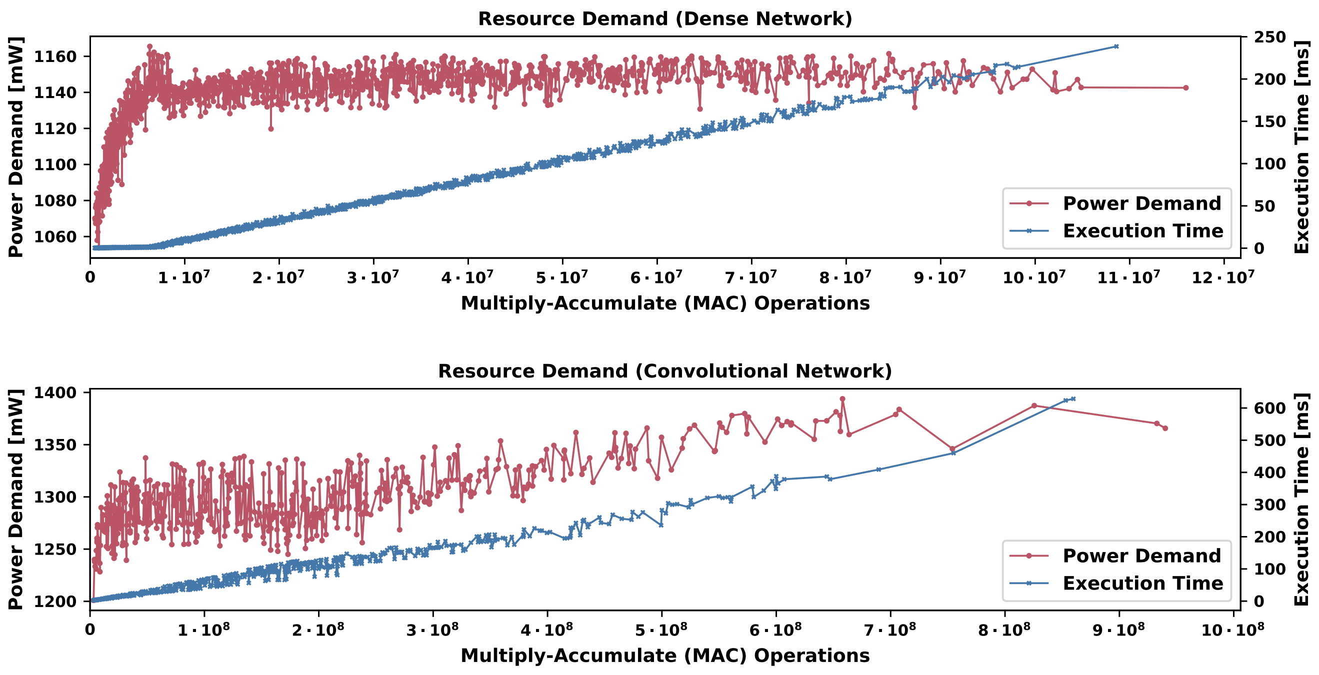

Figure 7: Influence of the number of MAC operations on the execution time and power draw for all generated dense (top) and convolutional (bottom) neural networks.

3 EVALUATION

As described in the previous section, one of the utilised input parameters for our resource-demand estimation models is the number of multiply-accumulate (MAC) operations. Figure 7 illustrates the power demand and execution time for all generated dense and convolutional networks. For both network types, the graphs show a clear linear relation between the number of MAC operations and the execution time. This is in line with the vendor's statement that these types of accelerators have a relatively deterministic run-time behaviour [11] and the number of MAC operations is a suitable metric to determine the execution time. The power demand also correlates to the number of MAC operations, but saturates at some point and no longer increases (dense networks) or increases only slowly (convolutional networks). We assume that at this point all hardware units are executing MAC operations and, thus, the TPU runs with full load and therefore with its maximum power demand.

We use 80 % of the generated and measured neural networks for the training of our resource-demand estimation models. Hence, the remaining 20 % can be used to evaluate our models. The evaluation set comprises 255 dense and 104 convolutional neural networks. For each neural network in the evaluation set, we measure the execution time and power draw, using the hardware setup as described in Section 2. Subsequently, we compare the measurements to the estimates given by the learned models, which have never encountered these networks during their training process. In summary, we learned two different resource-demand estimation models, that is, one model based on a linear regressor and one model based on random forest regressors. First, we present the results for the linear regressor and subsequently compare them with the results for the random forest regressors.

Figure 8 gives an overview of the estimation quality for convolutional neural networks. The graphs show the power demand (top) and execution time (bottom) as measured by our measurement setup and as estimated by our linear model. The graphs show that the estimates are close to the actual time and energy measurements over the whole range of considered neural networks. In particular, the mean absolute errors are 11.50 mW for the power demand and 4.25 ms for the execution time. The same holds for the estimations made for dense neural networks as illustrated in Figure 9. The only exception is the power demand for networks with a small number of MAC operations, which increases non-linearly where the linear model shows a linear increase and therefore introduces a corresponding, albeit small estimation error. However, the mean absolute errors are 6.77 mW and 0.17 ms for the power demand and the execution time, respectively. Thus, the linear model yields accurate results for the majority of networks.

In the next part of our evaluation, we repeat the estimations with a model based on random forest regressors. Figure 10 summarises the estimation quality based for the random forest model for convolutional neural networks. The mean absolute errors are 9.64 mW for the power demand and 4.42 ms for the execution time. As these estimations for power demands are between 1231 mW and 1394 mW we consider an absolute error of 9.64 mW very low. The same holds for the execution times, where the range is between 3 ms and 366 ms. For low values (around 3 ms) the model also estimates around 3 ms and the error of 4.42 ms primarily stems from

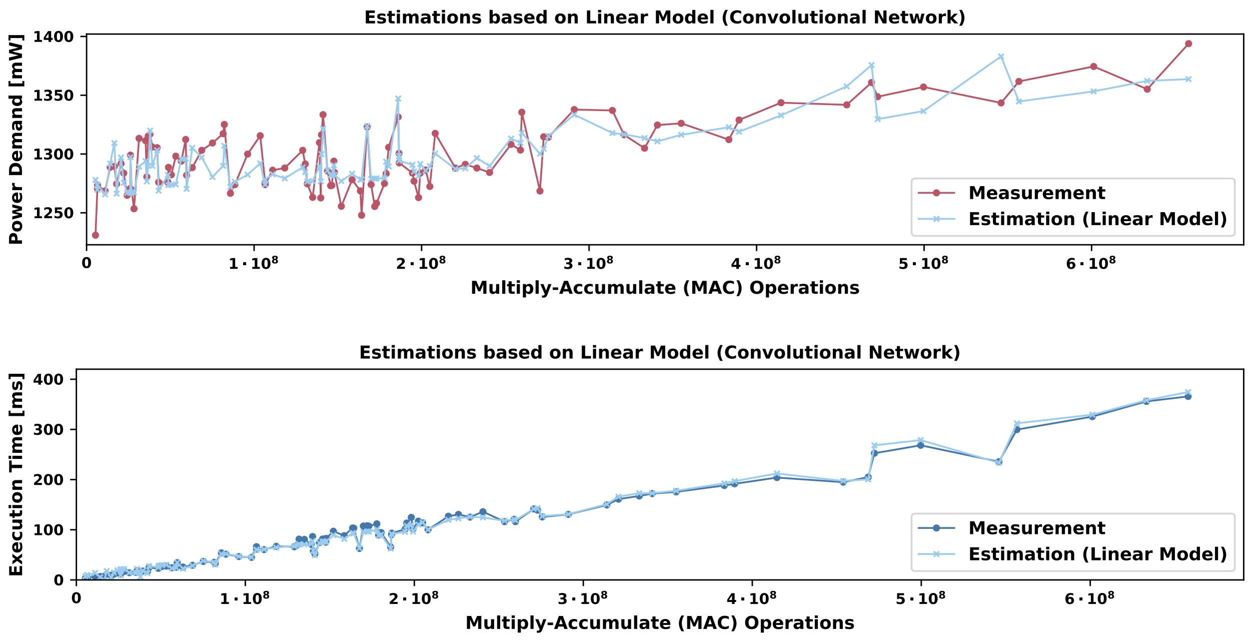

Figure 8: Measurements and estimations of the power demand (top) and execution time (bottom) for a convolutional network made by the linear model. The x-axis shows the corresponding number of multiply-accumulate (MAC) operations. The training data set and evaluation data set consist of 415 and 104 neural networks, respectively.

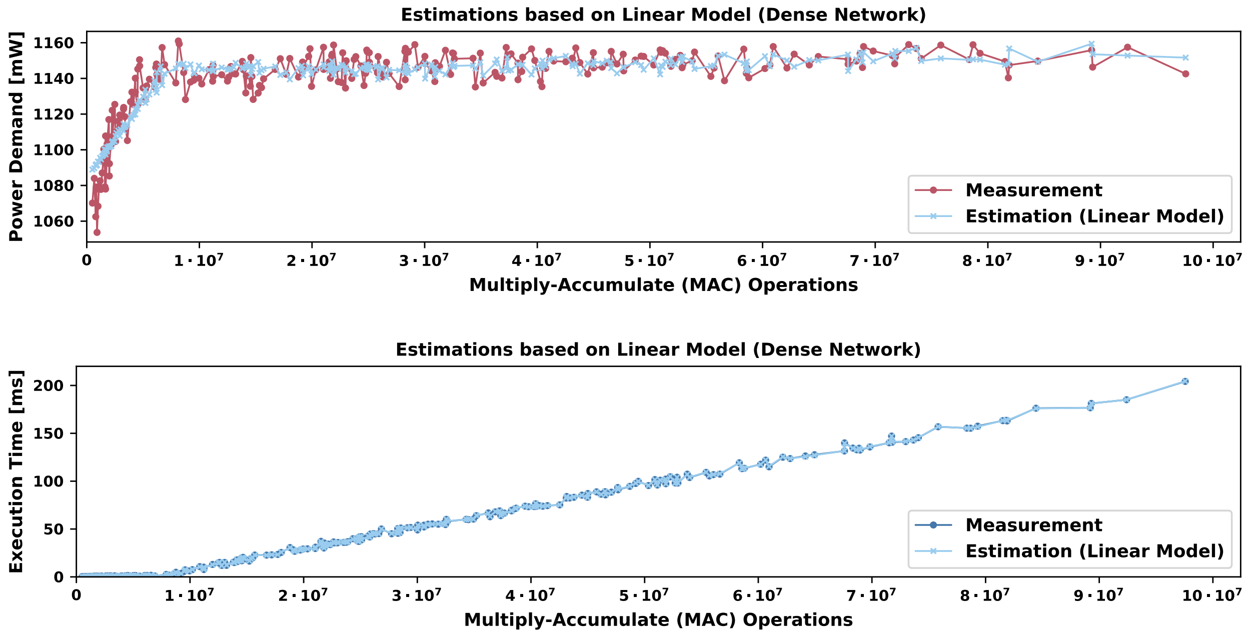

Figure 9: Measurements and estimations of the power demand (top) and execution time (bottom) for a dense network made by the linear model. The x-axis shows the corresponding number of multiply-accumulate (MAC) operations. The training data set and evaluation data set consist of 1016 and 255 neural networks, respectively.

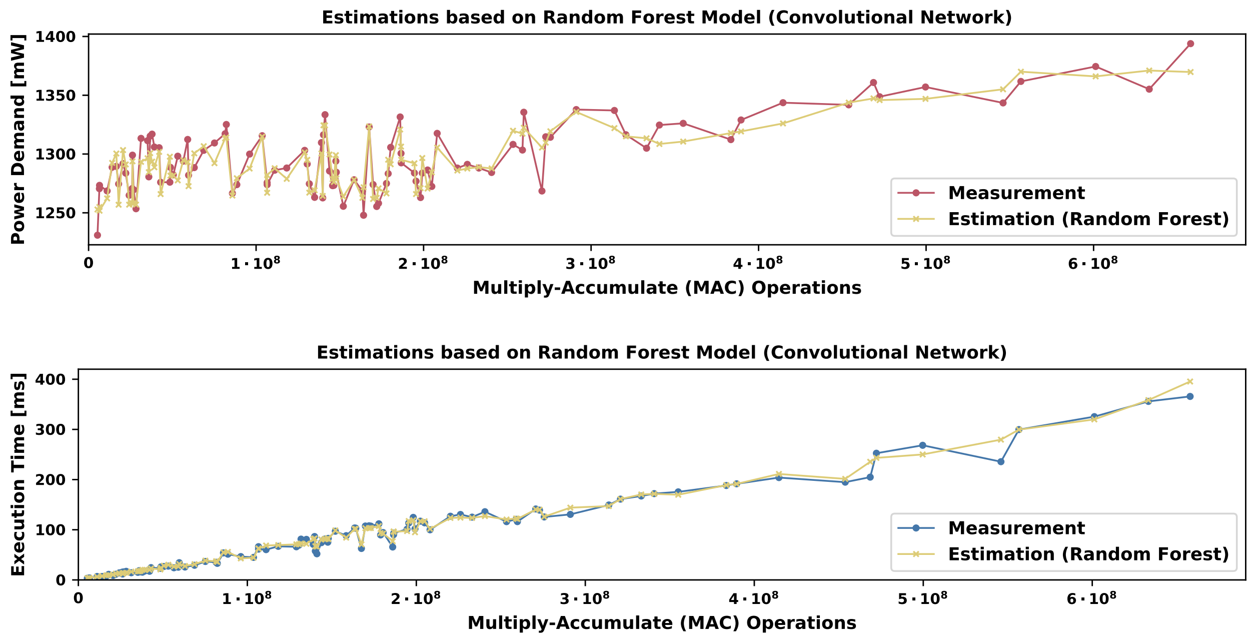

Figure 10: Measurements and estimations of the power demand (top) and execution time (bottom) for a convolutional network made by the random forest model. The x-axis shows the corresponding number of multiply-accumulate (MAC) operations. The training data set and evaluation data set consist of 415 and 104 neural networks, respectively.

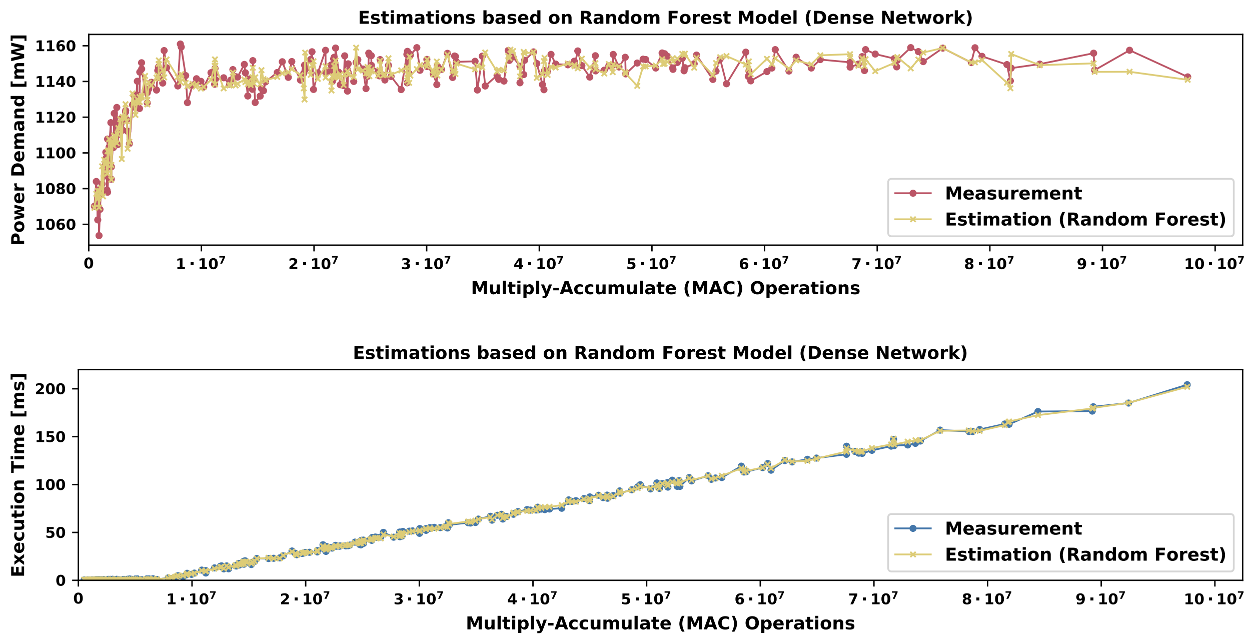

Figure 11: Measurements and estimations of the power demand (top) and execution time (bottom) for a dense network made by the random forest model. The x-axis shows the corresponding number of multiply-accumulate (MAC) operations. The training data set and evaluation data set consist of 1016 and 255 neural networks, respectively.

estimations for greater execution times. Hence, the estimations are relatively accurate for the complete range of values.

The results for dense neural networks are shown in Figure 11. Again, the results are relatively accurate. The mean absolute errors are 5.58 mW for the power draw and 0.73 ms for the execution time. These errors occur for values, which range from 1054 mW to 1161 mW for the power demand and around 0.3 ms to 204 ms for the execution time. Furthermore, the random forest model precisely models the non-linear increase in power demand for dense neural networks with a small amount of MAC operations. Thus, the estimation accuracy of the random forest model is similar for most neural networks compared to the linear model and overcomes the limitations of the linear model for small neural networks.

In summary, the estimations for dense networks are more accurate, compared to convolutional networks. The slightly increased training and configuration costs for the random forest model enable more accurate results. However, all estimations are close to the measured results for both models, and the variety between networks is much larger than the difference between measurements and estimations. Hence, our models allow developers an easy and accurate estimation of the resource demand and relieve them from time and energy measurements.

4 RELATED WORK

Previous work has thoroughly examined the power and performance characteristics of CPUs, GPUs, and TPUs [11, 16, 30] in data centres. Our work, in contrast, focuses on neural network execution on embedded platforms. In comparison to regular and large-scale systems, embedded platforms have a much lower power draw, and thus very different power-to-performance characteristics. Furthermore, the main focus for embedded systems is on response time rather than throughput [25].

Li et al. [16] compare the resource demand of training frameworks for convolutional neural networks on CPUs and GPUs. They further provide information on the effects of performance-tuning parameters, such as dynamic voltage and frequency scaling and hyper-threading, on the energy efficiency of the training processes. Our work presented in this paper, in comparison, targets hardware accelerators on embedded platforms.

Jouppi et al. [11] describe the architecture of a TPU deployed in data centres. They compare the performance and power demand to CPUs and GPUs, showing that the TPU outperforms both hardware alternatives. They also discuss that the execution model of the TPU is more deterministic, compared to CPUs and GPUs. The consequence is that the resource demand (in particular, power draw and response times) is much more predictable. Our measurements, in particular Figure 5 and 6, confirm the repeatable behaviour. However, access to the TPU presented in this paper is only available via cloud services.

Based on a cycle-accurate simulator, Gupta et al. [6] build a "latency predictor" for the Google Coral Edge TPU [22]. This latency predictor is then used together with the model accuracy to iteratively refine neural networks, until it achieves the desired prediction quality in the available response time budget. They further report that the TPU operates more power-efficiently when the model fits the on-chip memory, which is the case for the neural networks we have generated. Similarly, Kaufman et al. [12] model the execution time of tensor computations with a feed-forward neural network. In comparison, we demonstrate that even simple machine learning techniques, such as linear models, can estimate the resource demand for embedded accelerators adequately.

Sieh et al. [29] create execution-time and energy-demand models for an embedded microcontroller. Similar to our approach, they generate input programs automatically, measure the resource demand, and derive models. They formulate and solve an integer linear program (ILP) which yields the per-instruction costs of the examined hardware. Hönig et al. [7] use deep neural networks for energy models that also account for inter-instruction effects, for example, related to caches. TPUs, in comparison, avoid such inter-instruction effects [11]. Instead, tensor processing units aim at providing predictable execution times by exploiting data parallelism in hardware [4, 10]. Thus, high overall performance is achieved with a much more deterministic execution model, compared to CPUs. In consequence, the resource demand prediction of a neural network accelerator can work with simpler models.

5 FUTURE WORK

Future work will extend the resource demand models to support various additional neural networks architectures. In particular, more layer types (e.g., "deconv"), models with mixed layer types, activation functions other than "relu", and further features will be modelled. We plan to support more flexible convolutional layers with different input, output, and filter sizes. These extensions allow us to evaluate the resource demand of real-world applications.

Furthermore, Figure 4 shows the lifetime of the neural network on the embedded accelerator and indicates that the model execution cycle also comprises a pre- and a post-processing stage where the power draw is increased even though no inference is running. For embedded applications, modelling these stages can be important as well. Especially for applications with only few inferences, the transmission and pre- and post-processing stage constitutes a significant amount of the energy demand. Thus, the pre- and post-processing stages must be considered for a holistic resource-demand analysis.

Another important aspect to include in future models is the temperature. Since power draw and temperature influence each other, the accurate modelling of thermal effects is difficult. This is particularly important for the alternative support library version that enables a higher clock speed of the TPU (cf. Section 2.2), and thus causes increased temperatures. Besides the increased power draw, an accurate temperature estimation may also be important for the thermal design of real-world devices that include hardware accelerators for the execution of machine-learning workloads.

Figures 5 and 6 demonstrate that even the detailed power-over-time behaviour is repeatable amongst neural network executions. In consequence, precise modelling of this power curve could result in more accurate energy models. One interesting extension would be the inclusion of cycle-accurate simulation models built by accelerator vendors. We expect more accurate results compared to the current input features of PRECIOUS. However, to the best of our knowledge, there is no cycle-accurate simulation model publicly available for the TPU used in this work.

6 CONCLUSION

This paper has presented Precious, a system to estimate the resource demand of deep neural networks on embedded accelerator hardware. The resource demand estimation is based on actual measurements of power draw and execution time, using real hardware. From these measurements, Precious learns resource-demand models, using linear and random forest regressors. The resulting models show only small estimation errors, which means that the runtime behaviour of neural networks on dedicated hardware is well-predictable.

ACKNOWLEDGEMENT

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – project number 146371743 – TRR 89 "Invasive Computing", under grant no. SCHR 603/13-1 ("PAX"), and under grant no. SCHR 603/15-2 ("eLARN").

REFERENCES

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th Symposium on Operating Systems Design and Implementation (OSDI'16)*. USENIX, 265–283.

[2] Arash Ashari, Shirish Tatikonda, Matthias Boehm, Berthold Reinwald, Keith Campbell, John Keenleyside, and P. Sadayappan. 2015. On Optimizing Machine Learning Workloads via Kernel Fusion. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP'15)*. ACM Press, 173–182.

[3] Cody Coleman, Daniel Kang, Deepak Narayanan, Luigi Nardi, Tian Zhao, Jian Zhang, Peter Bailis, Kunle Olukotun, Chris Ré, and Matei Zaharia. 2019. Analysis of DAWNBench, a Time-to-Accuracy Machine Learning Performance Benchmark. *SIGOPS Operating Systems Review* 53, 1 (July 2019), 14–25.

[4] Jeff Dean, David Patterson, and Cliff Young. 2018. A New Golden Age in Computer Architecture: Empowering the Machine-Learning Revolution. *IEEE Micro* 38, 2 (March 2018), 21–29.

[5] Analog Devices. 2020. LTC2991. https://www.analog.com/en/products/ltc2991.html. Acc. 2020-02-20.

[6] Suyog Gupta and Mingxing Tan. 2019. EfficientNet-EdgeTPU: Creating Accelerator-Optimized Neural Networks with AutoML. https://ai.googleblog.com/2019/08/efficientnet-edgetpu-creating.html. Acc. 2020-02-20.

[7] Timo Hönig, Benedict Herzog, and Wolfgang Schröder-Preikschat. 2019. Energy-Demand Estimation of Embedded Devices Using Deep Artificial Neural Networks. In *Proceedings of the 34th Symposium on Applied Computing (SAC'19)*. ACM Press, 617–624.

[8] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. 2018. AI Benchmark: Running Deep Neural Networks on Android Smartphones. In *Proceedings of the Perceptual Image Restoration and Manipulation Workshop and Challenge (PIRM'18)*. Springer International Publishing, 288–314.

[9] Intel Inc. 2020. Intel Neural Compute Stick 2. https://software.intel.com/en-us/neural-compute-stick. Acc. 2020-02-20.

[10] Norman Jouppi, Cliff Young, Nishant Patil, and David Patterson. 2018. Motivation for and Evaluation of the First Tensor Processing Unit. *IEEE Micro* 38, 3 (May 2018), 10–19.

[11] Norman 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. 2017. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture (ISCA'17)*. ACM Press, 1–12.

[12] Samuel Kaufman, Phitchaya Phothilimtha, and Mike Burrows. 2019. Learned TPU Cost Model for XLA Tensor Programs. In *Proceedings of the Workshop on ML for Systems at NeurIPS 2019*. 1–6.

[13] Nicolas Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. 2017. Squeezing Deep Learning into Mobile and Embedded Devices. *IEEE Pervasive Computing* 16, 3 (July 2017), 82–88.

[14] Nicholas Lane and Petko Georgiev. 2015. Can Deep Learning Revolutionize Mobile Sensing?. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications (HotMobile'15)*. ACM Press, 117–122.

[15] Seyyed Salar Latifi Oskouei, Hossein Golestani, Matin Hashemi, and Soheil Ghiasi. 2016. CNNdroid: GPU-Accelerated Execution of Trained Deep Convolutional Neural Networks on Android. In *Proceedings of the 24th ACM International Conference on Multimedia (MM'16)*. ACM Press, 1201–1205.

[16] Da Li, Xinbo Chen, Michela Becchi, and Ziliang Zong. 2016. Evaluating the Energy Efficiency of Deep Convolutional Neural Networks on CPUs and GPUs. In *Proceedings of the International Conferences on Sustainable Computing and Communications (SustainCom'16)*. IEEE, 477–484.

[17] Google LLC. 2020. Core Layers. https://keras.io/layers/core/. Acc. 2020-02-20.

[18] Google LLC. 2020. Edge TPU Compiler. https://coral.ai/docs/edgetpu/compiler/. Acc. 2020-02-20.

[19] Google LLC. 2020. Get started with TensorFlow Lite. https://www.tensorflow.org/lite/guide/get_started. Acc. 2020-02-20.

[20] Google LLC. 2020. Get started with the USB Accelerator. https://coral.ai/docs/accelerator/get-started/. Acc. 2020-02-20.

[21] Google LLC. 2020. Tensorflow Keras. https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras. Acc. 2020-02-20.

[22] Google LLC. 2020. USB Accelerator. https://www.coral.ai/products/accelerator. Acc. 2020-02-20.

[23] Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, and et al. 2016. MLlib: Machine Learning in Apache Spark. *Journal of Machine Learning Researchs* 17, 1 (Jan. 2016), 1235–1241.

[24] A. Negi and P. K. Kumar. 2005. Applying Machine Learning Techniques to Improve Linux Process Scheduling. In *Proceedings of the TENCON 2005 - 2005 IEEE Region 10 Conference*.

[25] NVIDIA. 2015. *GPU-Based Deep Learning Inference: A Performance and Power Analysis*. Technical Report. NVIDIA Corp. 1–12 pages.

[26] scikit-learn developers. 2020. Linear Regressor. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html. Acc. 2020-02-20.

[27] scikit-learn developers. 2020. Random Forest Regressor. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html. Acc. 2020-02-20.

[28] scikit-learn developers. 2020. Randomized Search CV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html. Acc. 2020-02-20.

[29] Volkmar Sieh, Robert Burlacu, Timo Hönig, Heiko Janker, Phillip Raffeck, Peter Wägemann, and Wolfgang Schröder-Preikschat. 2017. An End-To-End Toolchain: From Automated Cost Modeling to Static WCET and WCEC Analysis. In *Proceedings of the 20th International Symposium on Real-Time Distributed Computing (ISORC'17)*. IEEE, 158–167.

[30] Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and Policy Considerations for Deep Learning in NLP. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL'19)*. 1–6.