Inference Energy Modeling for BERT Models

Chuning (Lily) Li

Department of Electrical & Computer Engineering

McGill University

Montréal, Québec, Canada

December 7, 2022

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of

Master of Science

Abstract

BERT achieved great success in Natural Language Processing (NLP). BERT models are computationally expensive due to their model size with hundreds of millions of parameters making them impractical to deploy to resource-constrained hardware platforms such as mobile devices. One way to optimize these complex models is to utilize hardware-aware neural architecture search (NAS). However, hardware-aware NAS needs to incorporate hardware performance metrics, such as energy consumption. Although on-device measurements provide accurate feedback, the overhead is huge. To address this problem, we propose an energy modelling framework on the Hikey 970 ARM big.LITTLE CPU and Mali GPU to predict the inference energy consumption of BERT models. We train energy predictor models on BERT and MobileBERT design space. We evaluate our BERT energy predictor's ability to generalize to the DistilBERT and DynaBERT design space Combined with the real accuracy values on the QNLI task, we evaluate our energy model's ability to predict Pareto-optimal front in the 2D accuracy-energy design space of DynaBERT models. Our model correctly predicts 14 out of the 17 true Pareto-optimal models.

Abrégé

BERT a obtenu un grand succès dans le traitement du langage naturel (NLP). Les modèles BERT sont coûteux en calcul en raison de leur taille de modèle avec des centaines de millions de paramètres, ce qui les rend peu pratiques à déployer sur des plates-formes matérielles à ressources limitées telles que les appareils mobiles. Une façon d'optimiser ces modèles complexes consiste à utiliser la recherche d'architecture neuronale (NAS) sensible au matériel. Cependant, les NAS compatibles avec le matériel doivent intégrer des mesures de performances matérielles, telles que la consommation d'énergie. Bien que les mesures sur l'appareil fournissent des informations précises, les frais généraux sont énormes. Pour résoudre ce problème, nous proposons un cadre de modélisation énergétique sur le processeur Hikey 970 ARM big.LITTLE et le GPU Mali pour prédire la consommation d'énergie d'inférence des modèles BERT. Nous formons des modèles de prédiction d'énergie sur l'espace de conception BERT et MobileBERT. Nous évaluons la capacité de notre prédicteur d'énergie BERT à se généraliser à l'espace de conception DistilBERT et DynaBERT Combiné avec les valeurs de précision réelles sur la tâche QNLI, nous évaluons

la capacité de notre modèle énergétique à prédire le front optimal de Pareto dans l'espace de conception 2D précision-énergie des modèles DynaBERT. Notre modèle prédit correctement 14 des 17 vrais modèles Pareto-optimaux.

Acknowledgements

First and foremost, I would like to express my gratitude towards my supervisor Professor Brett Meyer. He has been the kindest, most generous, empathetic and understanding. Without him, I would not have had the opportunity to start this research. I'm grateful for his guidance and support. I thank Seyyed Hasan Mozafari for providing constant advice and pulling me back on track. I highly appreciate his insightful feedbacks on my progress. I also thank my colleagues at McGill, especially Negin Firouzian, Mohamed Abdelgawad, Hung-yang Chang, Murray Kornelson, Hang Zhang, Mariham Amein who have helped me with my research.

Last but certainly not least, I would like to offer special thanks to my family and friends for their love and encouragement.

Contents

List of Figures

List of Tables

List of Acronyms

BERT	Bidirectional Encoder Representations from Transformers.
CPU	central processing unit.
FLOPS	floating point operation per second.
GPU	graphical processing unit.
MAPE	mean absolute percentage error.
NAS	neural architecture search.
NLP	natural language processing.

Chapter 1

Introduction

1.1 Deep Learning Models

Deep Neural Networks are now widely used to automate and solve complicated problems in various tasks. More specifically, in the area of Natural Language Processing (NLP), Transformers [4] have recently brought significant breakthroughs in many downstream tasks, especially with the introduction of BERT [1]. BERT achieved state-of-the-art performance on many common NLP tasks such as sentiment analysis and question and answering. In addition, BERT has been integrated into the Google Search engine to better understand queries.

1.2 Deep Learning Models on Embedded Devices

In the era of Internet of Things (IoT), embedded systems can support various applications in diverse domains: health, transportation, smart home, smart city, agriculture, education, etc. [5]. Pushing machine learning applications from cloud-dependent towards local processing provides many benefits: more reliability, lower latency, better privacy, and less power consumption [6]. First, local processing is more reliable since it does not require an Internet connection [7]. Second, locally generated data are not transmitted externally, therefore preserving user privacy [8]. Since no data is being transmitted, the delay of sending data to the cloud is avoided. Lastly, compared to cloud-based systems, algorithms running on embedded systems consume lower power and energy, and therefore, cause lower CO_2 emission [9].

1.3 Challenges of Deploying BERT on Embedded Devices

Despite the outstanding performance of Transformer-based models, such as BERT, the explosive size and computation cost makes it impractical to deploy these models onto resource-constrained devices, e.g., mobile devices. As an example, the BERT-base model consists of 110M parameters while the BERT-Large consists of 340M parameters [1]. On the other hand, embedded devices suffer from limited resources (energy, memory and

low-processing power). A representative hardware platform of smartphones such as Hikey 970 has a RAM size of 6GB operating at 12V DC power. Deploying large language models (BERT) in real-time applications on these battery-powered devices becomes a major challenge.

1.4 Optimization Techniques for BERT on Embedded Devices

There are outstanding research efforts proposed on optimizing and compressing BERT models for deployment on resource-constrained devices. These techniques can be categorized as : (1) pruning [10–14];(2) knowledge distillation [2, 15–20]; (3) quantization [21–25];(4) efficient network design [26, 27]. Models proposed by these works achieved competitive accuracy compared to the BERT-base model while significantly improving its computation cost. However, given different hardware platforms and efficiency constraints, designing specialized models to satisfy deployment constraints is computationally expensive and time-consuming. The design process needs to be repeated for each platform and hardware condition (battery condition, latency constraint), and the designed model needs to be retrained. It becomes impractical to design BERT models for each scenario [9].

1.5 Hardware-aware NAS

Neural Architecture Search (NAS) [28] is an effective solution for finding a set of optimal models. Taking into account hardware constraints (such as latency, power, energy, and memory footprint) and using multi-objective optimization algorithms in the search process gave rise to the sub-field of Hardware-aware NAS. A general NAS process contains 3 steps: (1) define a search space; (2) use a search strategy to sample a candidate model from the search space; (3) evaluate the candidate model and guide the search strategy based on the measured performance. The last two steps repeat until convergence or until reaching specified search steps. NAS has proven its ability to generate state-of-the-art models for Natural Language Processing tasks [29]. However, NAS is very computationally expensive and time-consuming due to the large design space and the training and evaluation of each candidate model. Especially for large language models such as BERT, training from scratch can take many GPU days. Therefore, it is important to find an efficient method to incorporate hardware constraints into the search process. Prior works propose three ways to incorporate hardware costs into NAS:

- **Proxy Metrics** [30, 31]: Use easy-to-obtain proxy metrics such as model size, and FLOPs [1] to represent latency or energy consumption. However, proxy metrics cannot guarantee a true representation. As an example, two models having the same number of FLOPs do not necessarily have the same latency [32]. In addition, the number of

[1] Floating-point operations per second

parameters or number of MACs are not good approximations for energy [33].

- **On-device Measurement** [34–37]: Use real-time measurements on the target device during the evaluation of candidate models. This method provides accurate hardware feedback, however, it significantly slows down the search process. The time required for on-device measurements of a large design space is prohibitive [38]. As an example, on-device energy measurement of a design space of 10^5 models would take 49 weeks on the Hikey970 platform (i.e., a representative hardware platform for smartphones).

- **Hardware Modeling** [39–41]: Using prediction models eliminates the overhead of real-time measurement. These methods are more efficient than on-device measurement and more accurate than proxy metrics. However, prior works just target latency as the hardware metric for BERT models. *To the best of our knowledge, this is the first work that targets energy consumption for BERT models on edge devices.*

1.6 Energy Modeling

Energy consumption is a vital consideration for power-hungry BERT models executing on edge devices. Mobile devices are battery-powered and therefore are constrained by finite energy. Although there exist some research works on latency modelling or power modelling techniques on edge devices [32, 39, 41–45], there's limited work on energy modeling for machine learning models. To the best of our knowledge, there's no work proposed on

energy modeling for BERT models on edge devices.

1.7 Thesis Statement

Hardware-aware NAS is a solution for finding a set of models that satisfies different hardware constraints. However, as discussed above, taking energy consumption into consideration is challenging due to the time cost of on-device measurement and accuracy. Building an energy predictor model is the most feasible solution among the three. In this thesis, we focus on the problem of building accurate energy predictor models for BERT inference on resource-constrained devices.

We establish an energy measurement framework based on previous works and our experiments. Our energy measurement framework takes 240s for obtaining measurement on a candidate model. We develop DNN models to predict the inference energy consumption of BERT models on the Hikey970 [46] CPU and GPU. We compare the mean absolute percentage error (MAPE) of our predictor with other regression techniques. In addition, since our predictor is designed to be used in Hardware-aware NAS, we compare the true Pareto-optimal front and the predicted Pareto-optimal front based on the predicted energy consumption and real accuracy data. We perform a case study on the DynaBERT design space with 240 models, with an accuracy metric evaluated on the QNLI task from the GLUE benchmark. We show that our model correctly predicts the models that are in the true Pareto-optimal front line in the DynaBERT design space on CPU.

1.8 Contribution

Our contributions are summarized in the following:

- **Energy Measurement:** We propose an energy measurement framework for obtaining accurate and stable system-wide energy consumption of running BERT inference on embedded devices such as Hikey970.
- **Energy Predictor:** We propose energy predictors for various configurations of BERT-like models on the Hikey970 CPU and GPU. Prior works require 1800 measurements [32], our model uses 800 models to build the energy model, which is 55% less on-device measurements.

An extended abstract and poster based on this research have been accepted at the Edge Intelligence Workshop 2022.

1.9 Thesis Organization

The thesis is organized as follows. We review past research efforts on energy modeling techniques in Chapter 2. Chapter 3 covers the fundamental knowledge on BERT, DistilBERT, and MobileBERT model architecture. In Chapter 4, we present our methodology of energy measurement and modeling. Chapter 5 discusses our experiments and results. In Chapter 6, we present a case study for NAS BERT (DynaBERT) [47]. Lastly, we discuss the future works in Chapter 7.

Chapter 2

Related Work

Predicting energy consumption on CPU and GPU is challenging due to the complexity of their hardware, memory hierarchies, and different parallelism schemes that they utilize. In the following, we categorize past works on energy prediction based on the approaches. As there's limited work on energy consumption and due to the close relationship between energy and power, we also review works focusing on power modeling. For each category of work, we discuss the advantages, drawbacks, and limitations.

2.1 Proxy Metrics

Prior works use easy-to-obtain metrics to represent energy consumption in searching for energy-efficient DNN models. Work by Song *et al.* [48] explores pruning for efficient convolutional neural networks (CNN). Experiments are carried out on Nvidia TitanX and

GTX980 GPUs. The evaluation of neural network energy consumption is done by using proxy metrics such as the number of weights and the number of FLOPs. Work in [30] uses the weighted sum of multiply-accumulate (MAC) operation cost and DRAM memory access cost to represent energy consumption. Work in [31] proposes an approach to design neural networks. They use a FLOP regularizer to prune the number of weights of a neural network. They use the number of FLOPs and the number of weights to represent hardware constraints: latency, and energy.

Disadvantages: Although these indirect metrics (MACs, FLOPs, number of weights) are easier to measure compared to on-device energy consumption, proxy metrics cannot guarantee a true representation of energy [33].

- Data movement has a greater impact on energy consumption than computation. As an example, accessing DRAM may cost 200x more energy than a MAC computation in an arithmetic logic unit (ALU) [49]. Therefore, the number of operations cannot represent energy consumption.

- Memory hierarchy and dataflow have a large impact on energy consumption in data movement. Energy consumption of the weights depends on their movement in different memory hierarchies, therefore, the number of weights of a DNN is not a good representation of energy.

How we address the limitations: We do not use proxy metrics to represent energy

consumption. Our predictor model is trained on real measurements collected on the target device.

2.2 On-device Measurement

Another category of work uses real-time measurements on target devices during the evaluation of candidate models to provide accurate hardware feedback.

Work in [36] proposes an algorithm called NetAdapt that automatically prunes pre-trained deep neural networks to satisfy given resource constraints on a mobile device. Hardware costs such as latency and energy are measured on the target device during the pruning process. The NetAdapt algorithm is guided by these direct measurements. They conducted experiments on image classification tasks on the ImageNet dataset [50]. The resulting network from adapting MobileNet [51] achieves 1.7x speedup and 0.3% accuracy increase.

Work in [35] proposes a hardware-aware NAS algorithm with reinforcement learning for energy-efficient CNNs. Their experiments are conducted on an Intel XEON E5-2620v4 processor equipped with GeForce GTX1080Ti GPU cards. They use the NVIDIA profiling tool, nvprof, to measure peak power, average power, and energy consumption on the target device. With 600 iterations in design space exploration, their algorithm demonstrates the ability to discover novel architectures with higher accuracy and lower energy than the ones designed by humans.

Work in [52] proposes a mixed-precision quantization framework to compress and accelerate DNNs inference. Instead of using indirect proxy metrics such as FLOPs or model size, direct energy consumption feedback is incorporated into the framework. Their framework effectively reduced latency by $1.4 - 1.95\times$ and energy consumption by $1.9\times$ with negligible loss in accuracy.

Disadvantages: Real-time on-device measurement results in accurate energy consumption, however, the overhead is huge. The overhead is caused by the following:

- (1) compilation: a model needs to be converted to a format that works on the target device (e.g., Tensorflow Lite).
- (2) inference: to obtain stable and accurate measurements, warm-up cycles are required. In addition, multiple runs of model inference are required to reduce the inherent variations in performance.
- (3) cool-down: continuous measurement on the target device could cause an increase in temperature, leading to incorrect measurements. Therefore, between each measurement, cool-down time is needed to decrease the temperature of the target device.

Previous works target DNN and CNN models that are significantly smaller and less complex than BERT models, e.g., ResNet [53] with 1202 layers has 19.4M parameters and BERT-base model contains 110M parameters [1]. The design space of BERT models is large

and therefore requires significantly more time for on-device measurement. BERT model with 5 architectural parameters and each of them with 10 choices results in a design space of 10^5 models. The time required for taking energy measurements of BERT models is huge. Taking energy measurement of a candidate model takes approximately 5 minutes. On-device energy measurement of a design space of 10^5 models would take 49 weeks on the Hikey 970 platform. The huge overhead of on-device measurement makes it impractical and prohibitive to employ in hardware-aware NAS.

How we address the limitations: Instead of using real-time on-device measurements, we develop energy predictor models that can be used in hardware-aware NAS. The design space we used to train our predictors consists of at most 800 models, which is significantly smaller than the design space of BERT models. We only measure energy consumption for training our predictor instead of incorporating on-device measurement in NAS. Therefore, to develop an energy predictor (our method), we require less time and computational power.

2.3 Operator-based Predictor

This type of work estimates energy consumption from the bottom up. The overall energy consumption is estimated as the sum of the energy consumption of smaller modules. The granularity of the modules may vary depending on the approach. Typically the cost of each type of operation is measured and the overall energy consumption is the weighted sum of operations and cost per operation.

Work in [54] proposes McPAT, an integrated power, area and timing modeling framework for multicore processors that provide estimations for dynamic power, static power and short-circuit power dissipation. The key component of the framework contains an XML interface which allows users to specify processor parameters and a performance simulator. McPAT takes a hierarchical approach to decompose modules to the circuit level to model the dynamic power, static power and leakage power. The framework is beneficial in terms of flexibility due to its provided XML interface, however, McPAT also has significant error caused by abstraction. In the work of Rethinagiri [55], compared with real board measurement on a dual-core ARM Cortex-A9 processor, McPAT has an average error of 23%.

In 2016, Chen *et al.* propose Eyeriss, an accelerator for CNNs [56]. They examined power breakdowns per layer. Results show that power consumption in ALU only accounts for 10% of the per-layer power consumption, which confirms that data movement consumes more than computation. Based on their observation, in 2017, Yang *et al.* proposed energy-aware pruning algorithms for CNNs [57]. Their energy estimation methodology is based on Eyeriss. Energy consumption is the sum of two parts: computation energy consumption and data movement energy consumption. Data movement energy consumption is calculated based on the number of memory accesses at each level of the memory hierarchy multiplied by the energy number of memory access. Computation energy consumption is calculated by counting the number of Multiply-and-Accumulate(MAC) operations multiplied by the energy consumed by each MAC operation. The energy number of memory access and MAC

operation are obtained from real measurements.

In 2017, Yoon *et al.* propose a practical and extensive power model for modern mobile processors that consist of diverse subsystem components [58]. The total power consumption is the sum of the power consumed by each component. The total power consumption is the sum of the power consumed by each component. The multicore CPU power consumption is the summation of the power consumption by the activated core and the uncore (base power of the microprocessor). The CPU power is modelled as a coefficient multiplied by utilization plus the base power consumption. The power model is evaluated on two types of experiments: benchmark tests and real-world applications. The average TEE(total energy error) and the average MAPE(mean absolute percentage error) of the real-world applications are 2.45% and 5.09%. This type of utilization-based bottom-up methodology is easy to implement on any type of device, however, the drawback is the accuracy because the power consumption varies greatly even with the same utilization.

Work in [59] proposes a quantization method with dynamic bit-level fusion/decomposition to accelerate DNNs. They developed a cycle-accurate simulator that takes instructions for the given DNN and simulates the execution to calculate the cycle counts as well as the number of accesses to on-chip buffers and off-chip memory. The architecture-level energy consumption is calculated by aggregating the energy consumption of each component in the system.

Work in [37] proposes a hardware-aware NAS algorithm with quantization to find energy-

efficient DNNs. They used the simulator proposed in [59] to obtain hardware performance metrics including energy consumption.

Marchisio *et al.* propose NAScaps [40], an automated framework for the hardware-aware NAS of different types of DNNs, covering both traditional convolutional DNNs and CapsNets. The hardware consumption such as energy and latency are modelled by accumulating elementary operations' cost (number of cycles, energy cost per layer). To obtain the elementary operations' cost, the hardware platform is described at the RTL level. Latency, energy, and memory costs of elementary operations are measured using VLSI CAD tools. The cost of each operation is then multiplied by the occurrences of the operation in the DNN model to obtain the total cost. The energy consumption is calculated based on memory accesses, the number of clock cycles, the number of weights and latency. Since the objective of the work is to find efficient DNNs, the accuracy of the hardware consumption modeling is not reported in the work, but the results of their NAS algorithm show that they are able to find high-accuracy networks with less energy and latency.

Disadvantages: In this category, typically, simulators are used to calculate cycle counts and memory accesses. This type of method requires in-depth knowledge of the target device hardware. In addition, parallelism in the hardware and complex memory hierarchies prevent these types of methods to achieve high accuracy.

How we address the limitations: We do not use simulators to model energy

consumption. All data are collected on the actual target device, therefore hardware characteristics are captured more accurately.

2.4 Performance-counter-based Predictor

Different from bottom-up methods presented in the previous section, this category of works estimates energy from the top-down. Usually, parameters such as performance counter values are collected and used to train a regression model to predict energy consumption.

In 2013, Pricopi *et al.* uses the static program analysis method to model the performance of individual cores and developed performance and power models across different cores for the ARM big.LITTLE architecture [60]. This method overcomes the challenges of modelling and estimation of two significantly different cores. They first developed the CPI(cycles per instruction) stack-based performance model for each core, then used the CPI value along with additional information, such as instruction mix, memory behaviour and so on, to estimate the power behaviour. The power consumption is expressed as a linear regression model. This method could derive power estimation of the second core based solely on the execution profile of the first core, hence eliminating the cost of executing the application on the second core. They evaluated the power model in terms of fitting error, intra-core validation and inter-core validation. The average prediction error is 3.9% for inter-core validation. Although the prediction error is low, the power model is built on top of the performance model, in addition to this technique, requires conducting offline analysis on the target platform to determine

the events to monitor. Another drawback is that the measured power across the training benchmarks suggests that the minimum and maximum power consumption of the small core is small, therefore the power consumption on the small core is not modeled. However, in our experiments, we identify that the Hikey 970 CPU small cores are different from what they observed.

In 2016, Walker *et al.* build an accurate and stable run-time power model using performance monitoring counters (PMCs), CPU clock frequency and CPU voltage for mobile and embedded device [61]. The platform used in this work is an ARM big-LITTLE platform. This work shows that a set of PMC events should be carefully chosen to provide the maximum amount of information and stability of power consumption. They used statistical analysis on all the PMC events for each CPU cluster and provided an automated PMC event selection methodology. Since the power consumption on CPU consists of static power and dynamic power, they propose to calculate these power separately. The static power is calculated based on the clock frequency and CPU voltage, whereas the dynamic power, in addition, also depends on the PMC events. As a result, the power consumption is modelled as a linear regression model and it achieves 3.8% and 2.8% average error on ARM Cortex-A7 and Cortex-A15, respectively. The drawback of any power modeling technique relying on hardware performance counters is the limitation of hardware architecture dependency, as the hardware performance counters available for different hardware vary.

In 2017, Zhang *et al.* propose an idle-state-based CPU power modelling method to

estimate the power of multi-core smartphones more accurately [62]. In fact, they stated that idle states of CPUs and the duration that a task is running on a specific CPU affect the power of the multi-core system in addition to workload and frequency. According to their findings, larger computation duration will face a drop in power consumption with the fixed frequency and latency, which is caused by deeper idle states. Therefore they have considered both duration of idle states and the number of entries for idle states to generate a more accurate power model of multi-core mobile systems. Their approach was able to achieve a high average accuracy of 98% for various benchmarks, and 96% for real applications.

In 2017, Cai *et al.* propose NeuralPower, a polynomial regression model-based prediction framework for layer-wise power consumption for CNNs running inference on GPU [63]. They model layer-level power consumption as the sum of two components. The first component is a polynomial term based on the architectural parameters of a layer such as a kernel size, stride size and padding size. The second component is a polynomial term based on special terms which are different from one layer type to another including the number of memory access, the number of floating point operations. Similar to the power consumption model, they also built a runtime model to predict layer-wise latency. Combining the two models, the energy consumption is the scalar product of the power and runtime prediction vectors. Experiments are conducted on the Nvidia GeForce GTX Titan X. As a result, their models achieve an average accuracy of 88.24% in runtime, 88.34% in power, and 97.21% in energy.

In 2018, Rodrigues *et al.* build a performance counter-based prediction framework for

modeling CNN per-layer energy consumption [64]. They used a simple multi-variate linear regression model based on the number of SIMD instructions executed and the number of bus accesses to estimate energy consumption. The embedded system chosen is conducted on an NVIDIA Jetson Tegra X1 board on single-core ARM Core A57. Energy measurements are obtained by using ARM streamline analyzer running various configurations of convolutional layers. A subset of the obtained energy measurements are used as the training set, leaving a small set for cross-validation. With both the SIMD instructions count and the number of bus accesses, their prediction model achieves an average relative test error of approximately 8%.

Disadvantages: Using linear regression models inherently assumes a linear relationship between input features and the output. However, the linearity between the performance counters and the energy consumption is not proven. Aside from the regression model, using performance counters have the following drawbacks and limitations:

- The limit on the number of simultaneously collected counters. As discussed in [65] where they used 13 performance counters, the number of simultaneously monitored counters is limited to 4, therefore it requires multiple runs to obtain a single training sample.

- For GPU, the counters record values per streaming multiprocessor instead of the whole GPU which is not a direct indication of the real values.

- Some important performance counters are not available on all target devices such as the DRAM data read counter.

How we address the limitations: To address the aforementioned limitations, we choose DNN with architectural parameters as input features to predict energy consumption. DNN better captures non-linearity between input features and the output. In addition, we use only architectural parameters instead of performance counter data.

Chapter 3

Background

Pre-trained language models such as BERT (Bidirectional Encoder Representations from Transformers) [1] achieved great success in Natural Language Processing (NLP). BERT is a Transformer-based model with the stacking of multiple bidirectional Transformer encoder layers. Training of a BERT model requires two steps: pre-training and fine-tuning. BERT is first pre-trained on unlabeled data with two tasks: Masked Language Modeling and Next Sentence Prediction. It is then fine-tuned on labelled data for specific downstream tasks. The BERT-base model contains over 110M parameters. Its heavy model size makes it impractical to be deployed to resource-constrained platforms. Recent works [2, 16, 26, 27] are proposed to build lightweight BERT-like models that suffer little from a drop in accuracy. In the following, we discuss the architecture of the BERT-base model, and two light-weighted BERT models: DistilBERT [16] and MobileBERT [2].

3.1 BERT Model Architecture

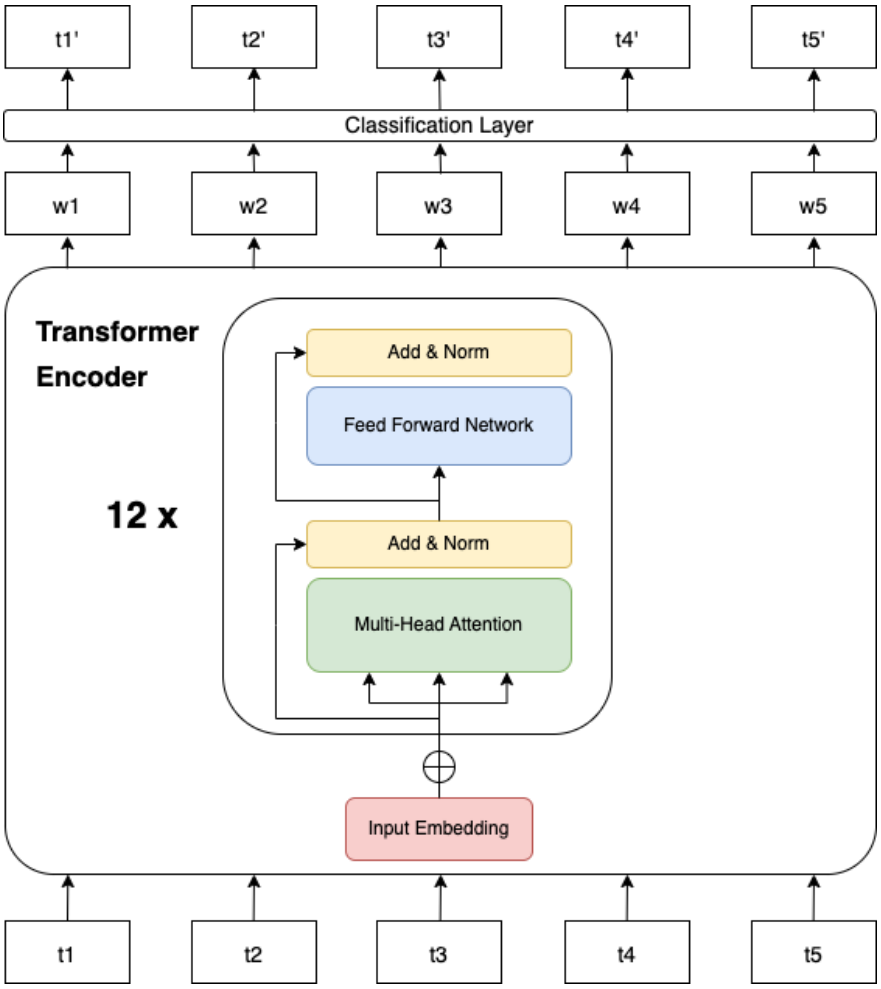

Figure 3.1: BERT architecture

BERT [1] is based on the Transformer [4] model, consisting of a stack of transformer encoders. Figure 3.1 shows the architecture of the BERT-base model. The BERT-base model consists of an embedding layer and a stack of 12 transformer encoder layers. Input data is fed into the embedding layer, and the output passes through each encoder layer

sequentially. Each of the encoder layers consists of a Multi-Head Attention(MHA) block and a Feed-Forward Network(FFN) block.

3.1.1 Embedding Layer

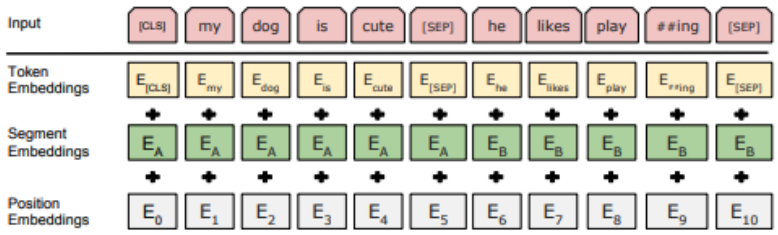

Figure 3.2: BERT input representation [1]

Figure 3.1 shows the embedding layer operations of BERT. The input sequence is padded to a fixed sequence length S. Each word in an input sequence is embedded into a token using WordPiece embeddings [66] with an embedding size denoted as H. For each input sequence, a positional embedding is used to indicate the position of each token. A segment embedding is also added to identify which sentence a token belongs to. As a result, the output of the embedding layer is the summation of the three embeddings. Increasing S and H increases the number of matrix addition options needed to perform in the embedding layer.

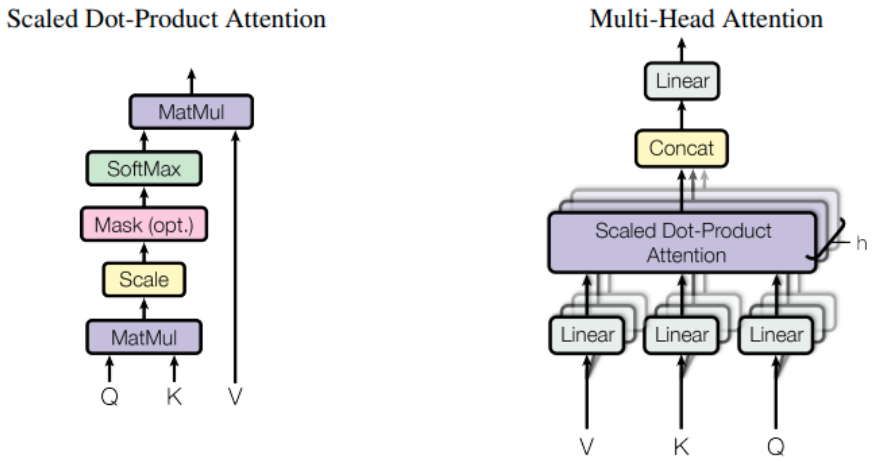

Figure 3.3: Multi-Head Attention [1]

3.1.2 Multi-Head Attention

In the MHA block, each attention head mechanism consists of a set of Query, Key and Value weight matrices. Attention head mechanism, as shown in Figure 3.3, allows the model to determine how closely two words in a sequence relate to each other. This is achieved by the following: First, multiplying the embedding vector with these three matrices generates a query vector, a key vector and a value vector. Secondly, taking the dot product of each pair of the key vector and query vector and normalizing it by taking softmax. Lastly, multiplying the softmax score with each of the value vectors and sum up the weighted value vectors. With multiple attention heads, since there's no data dependency, the calculation can be done in parallel as shown in Figure 3.3. Output from each attention mechanism is concatenated linearly and then passed to the FFN block.

We denote the number of attention heads as A. With more attention heads, more

query, key and value vector are added to the architecture and more matrix multiplications operations are performed. However, since the attention calculations can be done in parallel, the latency of the MHA block may not be affected by adding more attention heads.

3.1.3 Feed-Forward Network

The FFN block consists of 2 fully connected layers which perform 2 linear transformations where the intermediate size is denoted as I. Increasing I results in an increase of the weight matrix and more multiply and accumulation operations (MACs) in matrix multiplication. Modifying I directly affects the latency of BERT inference.

In conclusion, BERT model architecture can be described with 5 parameters: embedding size (H), sequence length (S), number of attention heads (A), number of layers (L), and intermediate size of the FFN (I). Table 3.1 presents the architectural parameters and their descriptions of the BERT-base model.

Table 3.1: BERT architecture parameter

Architectural parameter	**Description**
H=768	The dimension of the embedding layer/ the size of the embedding vector.
S=512	The sequence length of the input.
L=12	The number of encoder layers.
A=12	The number of attention heads in MHA block.
I=3072	The dimension of FFN.

3.2 DistilBERT

The explosive size of BERT makes it impractical to deploy to resource-constrained devices. DistilBERT [16] is proposed as a promising solution to reduce the size of the BERT model. DistilBERT halves the depth of the BERT-base model by making it a stack of 6 encoder layers instead of 12. By applying knowledge distillation techniques on BERT, DistilBERT is trained to reproduce the behaviour of BERT model. DistilBERT consists of 40% fewer parameters compared to BERT-base while retaining 95% of the performance. The score of BERT-base on GLUE tasks is 78.0, and DistilBERT achieves a score of 75.2. In addition, inference of DistilBERT is 60% faster than BERT-base.

3.3 MobileBERT

MobileBERT is a compact transformer-based model proposed to compress and accelerate BERT model. The architecture of MobileBERT [2] model is deeper and narrower compared to the BERT-base model. It consists of 24 encoder layers with just 25M parameters due to the reduced size of each building block. The architecture of MobileBERT model is shown in Figure 3.4.

MobileBERT architecture differs from BERT model in 2 main aspects. Firstly, it is equipped with bottleneck structures which are two linear transformations added before MHA block and after FFN block. The linear transformation before the MHA block

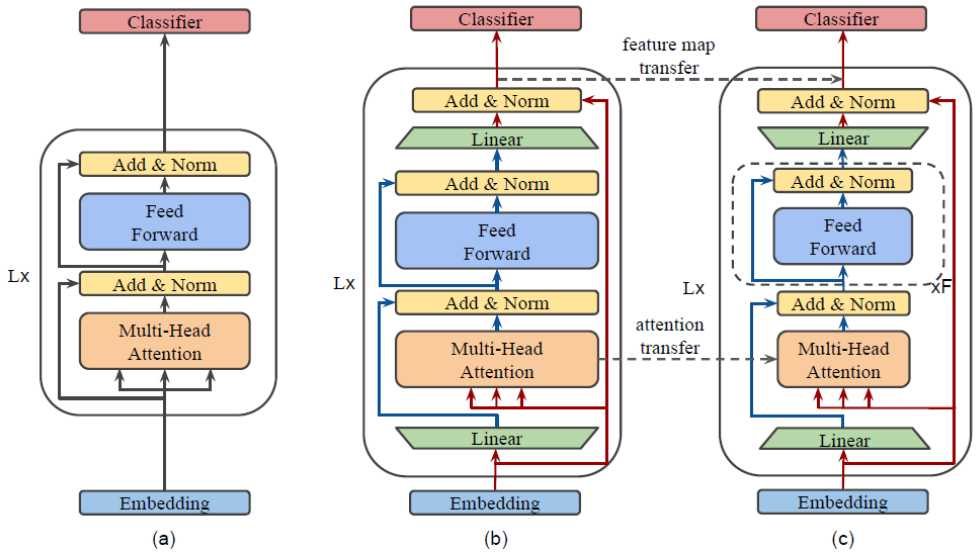

Figure 3.4: (a) BERT architecture (b) Inverted-Bottleneck BERT (IB-BERT) (c) MobileBERT architecture [2]

projects embedding from size 512 to size 128. Whereas the linear transformation after the FFN block projects the input from size 128 to size 512.

Secondly, due to the addition of the bottleneck structures, the ratio of MHA size and FFN size is not the same as in BERT-base model. The ratio of the MHA block and the FFN block in BERT-base model is 1:2. However, the input to the MHA block is the output from embedding layers which has a size of 512, while the input to the FFN block is the output from the bottleneck transformation with a size of 128. Therefore the MHA block results in more parameters than the FFN block. To address this problem, MobileBERT uses a stack of 4 FFN blocks to re-balance the size of MHA and FFN. With the addition of more FFN blocks, more matrix multiplications are performed in the computation.

To train the MobileBERT model, an Inverted-Bottleneck BERT model is first trained

to be used as a teacher model. The IB-BERT model has the same architecture as BERT-large with added inverted-bottleneck structure. MobileBERT is trained by applying knowledge distillation to transfer knowledge from the teacher model. With the addition of bottleneck layers in MobileBERT, three important architectural parameters are added to the architecture. Table 3.2 shows the architectural parameters of MobileBERT.

Table 3.2: MobileBERT architecture parameter

Architectural parameter	**Description**
H=512	The dimension of encoder layer.
S=512	The sequence length of the input.
L=24	The number of encoder layers.
A=4	The number of attention heads in MHA block.
I=512	The dimension of FFN.
E=128	The size of the embedding vector.
B=128	The size of the bottleneck layer output.
F=4	The number of FFNs.

Chapter 4

Methodology

The goal of our work is to predict inference energy consumption of BERT models with minimum amount of measurements. In order to model energy consumption, we need accurate energy measurements. Based on previous research and experiments [67–70], we designed our energy measurement framework that takes 240s to measure a candidate model. With a large design space of models, the time required for energy measurement is huge. Therefore, we need to carefully find the representative design space. In this regard, we examine the effect of each architectural parameters on BERT inference energy consumption. We consider different representative design spaces for different models and hardware (embedded CPU and GPU). We compared different regression techniques for energy modeling where we find that DNN achieves the best performances. Also, we explore the accuracy of our energy modeling in the 2D design space of latency and energy consumption. In terms of evaluation of predicting the

POFs in this 2D design space, we use a fast non-dominated sorting algorithm (NSGA-II [71]).

Our approach differs from prior works [61, 64, 72] in two aspects: (1) we use DNN instead of linear models, (2) we use BERT architectural parameters instead of Performance Monitoring Counters (PMCs) data. Using a linear regression model inherently assumes a linear relationship between input features and the output. However, this assumption is not proven in energy modelling [73]. Some research works use DNN [73, 74] with PMCs data as input features. Although these methods achieve higher accuracy in prediction, using PMCs is inefficient and sometimes infeasible due to the limitation of hardware architecture dependency, as the PMCs available for different hardware vary. Therefore, we choose DNN with architectural parameters as input features to predict energy consumption.

4.1 Design Space Analysis

We first examine the effects of each BERT architectural parameter on the inference energy consumption. With the knowledge of how each architectural parameter contributes to energy consumption change, we can construct a representative design space with fewer models by focusing more on parameters that have a larger effect on energy consumption, and eliminating parameters that have little effect on energy consumption. Understanding the design space allows us to limit the number of on-device measurements which turns in the reduction of computation time.

4.1.1 BERT Architectural Parameter Effects on Inference Energy Consumption on CPU

BERT-base model can be defined with five architectural parameters: embedding size (H), sequence length (S), number of attention heads (A), number of layers (L), and intermediate size of the FFN (I). Figure 4.1 shows the relationship between energy and each architectural parameter. For each of the sub-figures, we keep the other four architectural parameters to a fixed value and vary the target architectural parameter to observe its effect on energy consumption on CPU. As an example, in the first sub-figure with the x-axis labelled as Embedding size (H), we vary the embedding size in the range $H = \{128, 256, 384, 512, 640, 768\}$, and set the other parameters as the following: $S = 512, L = 4, A = 8, I = 512$.

We find that each of the architectural parameters has different effects on the energy consumption on CPU. First, sub-figure presents the effect of changing the embedding size (H) in the range $H = \{128, 256, 384, 512, 640, 768\}$. We observe a linear or exponential trend in energy consumption.

Changing the embedding size causes about an 80% change in energy consumption. Embedding size directly affects the input sequence size. With an increase in the embedding size, operations such as matrix multiplications in the embedding layer, MHA block and FFN block increase due to the increase of input. Therefore, energy consumption is significantly affected. In the second sub-figure, we observe that changing the number of

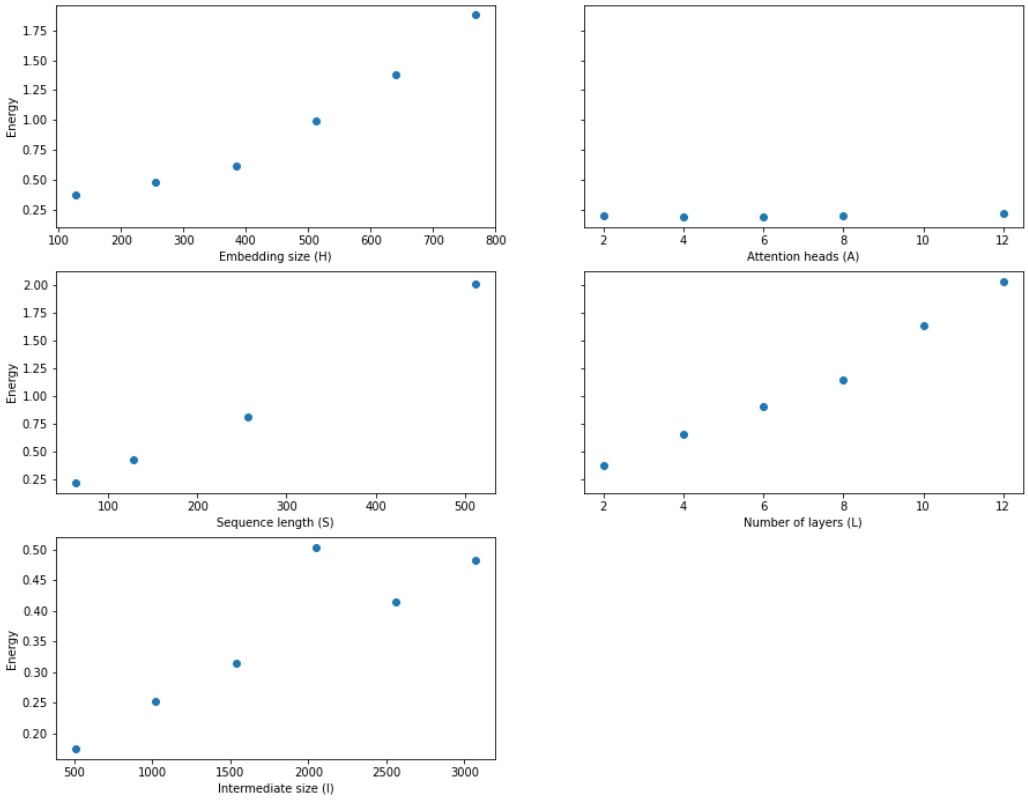

Figure 4.1: Energy consumption on CPU vs. BERT architectural parameters

attention heads contributes to less than 5% change in energy consumption. Among the 5 architectural parameters, the number of attention heads has the least effect on energy consumption. To further demonstrate this observation, we performed experiments with 3 different baseline models as presented in Figure 4.2. The configuration of the other 4 architectural parameters is presented in the figure. We conclude that under different

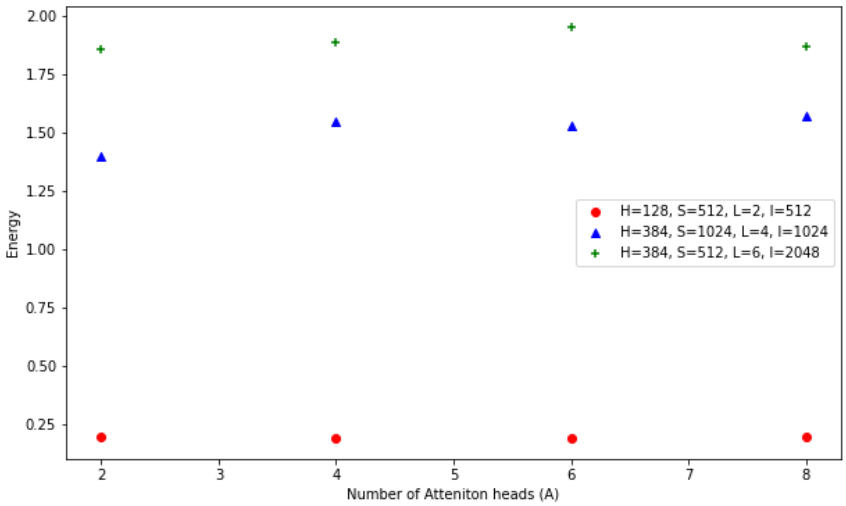

Figure 4.2: Energy consumption on CPU vs. Number of Attention heads (A)

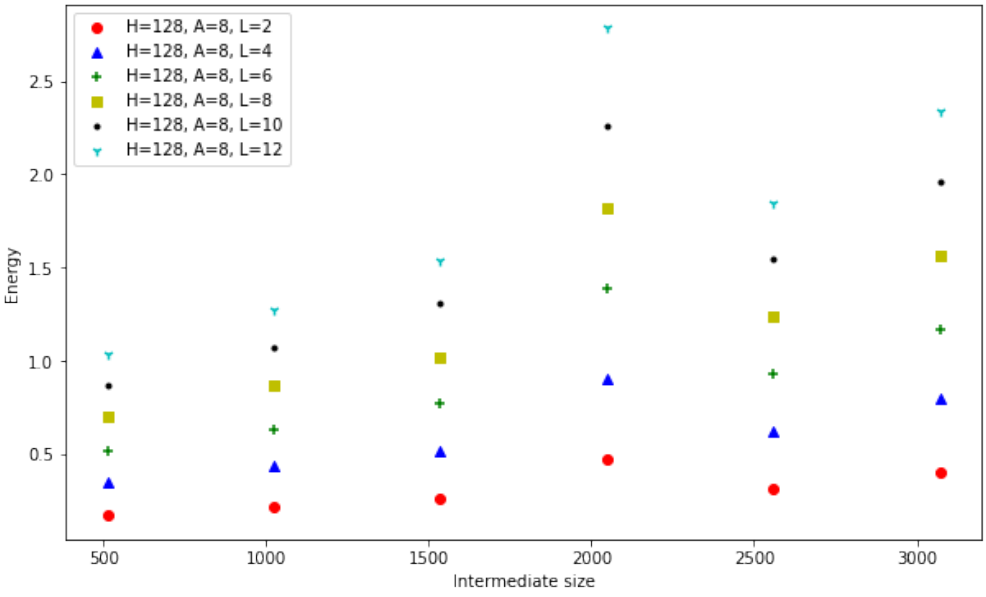

Figure 4.3: Energy consumption on CPU vs. Intermediate size (I)

settings, the number of attention heads (A) contributes very little to the change of energy consumption on CPU, which is at most 10%. We believe this is due the fact that the attention computations can be done in parallel, and therefore, the change in system active power is small. With the other architectural parameters set to the same value, the change in latency is also small. Since energy consumption is a product between latency and power, the change in energy consumption is small. To reduce the design space, we set the number of attention heads (A) to a fixed value of 8 for all models.

In terms of Sequence length (S) and Number of layers (L), these two parameters have similar effects on energy consumption. We vary the sequence length (S) in the range $S = \{64, 128, 256, 512\}$. We can observe that changing the sequence length has a large contribution to energy consumption on CPU. We vary the number of layers (L) in the range $L = \{2, 4, 6, 8, 10, 12\}$. We observe a linear relationship between the number of layers (L) and energy consumption.

Lastly, we experiment with different Intermediate sizes (I) in the range $I = \{512, 1024, 1536, 2048, 2560, 3072\}$. The relationship between the energy consumption and the intermediate size is more unpredictable compared to the other parameters. We observe that the energy consumption with $I = 2048$ is the highest. To demonstrate that this effect takes place in other baseline settings, we ran more experiments as presented in Figure 4.3. As an example, the red dots in Figure 4.3 are models with $H = 128, L = 2, A = 8, S = 512$ and different intermediate sizes (S). This observation

demonstrates that the relationship between intermediate size and energy consumption on CPU is non-linear. We believe using DNN as the energy predictor model better captures this relationship.

4.1.2 BERT Architectural Parameter Selections on CPU

We define a design space of BERT models by varying four architectural parameter values (H, S, L, and I) based on observations presented in section 4.1.1. In the literature, models targeting edge devices [2, 16–18, 26, 27, 45, 75] are smaller than BERT-base due to limited resources. Therefore, the upper bound of our design space is BERT-base (110M parameters) [1]. The lower bound of our design space is BERT-tiny (4M parameters), the smallest BERT model in the literature [18]. The candidate values of each architectural parameter are presented in Table 4.1. In this selection,

- The minimum and maximum value of H is set by boundary models of our design space. The values in between are chosen in accordance with the studies in [18] with the addition of 384 and 640.
- The lower bound of S is set to 64 based on [1] and the upper bound is 1024.
- L is set in accordance with the studies in [18].
- A is set to 8 as H is constrained to be a multiple of A.
- In [1, 18], S is set to 4H. We allow S to take the values from 4×128 to 4×768.

Models from the design space can take any combination of these 5 parameter values. The resulting design space is composed of 1080 models.

Table 4.1: BERT design space on CPU

Architectural parameter	Values
H	128,256,384,512,640,768
S	64,128,256,512,1024
L	2,4,6,8,10,12
A	8
I	512,1024,1536,2048,2560,3072

4.1.3 BERT Architectural Parameter Effects on Inference Energy Consumption on GPU

We perform experiments on the Mali GPU [76] (an embedded ARM GPU) to understand the effects of each architectural parameter on inference energy consumption shown in Figure 4.4. Unlike what we have observed on CPU, all the architectural parameters contribute to the change in energy consumption. Varying number of attention heads (A) from 2 to 12 results in an increase of inference energy consumption from 0.048 to 0.073. As a result, we must consider all of the architectural parameters in defining the design space.

4.1.4 BERT Architectural Parameter Selections on GPU

We define the design space of BERT models on GPU based on the observations presented in section 4.1.3. Since all 5 architectural parameters (H, S, L, A, I) affects energy consumption,

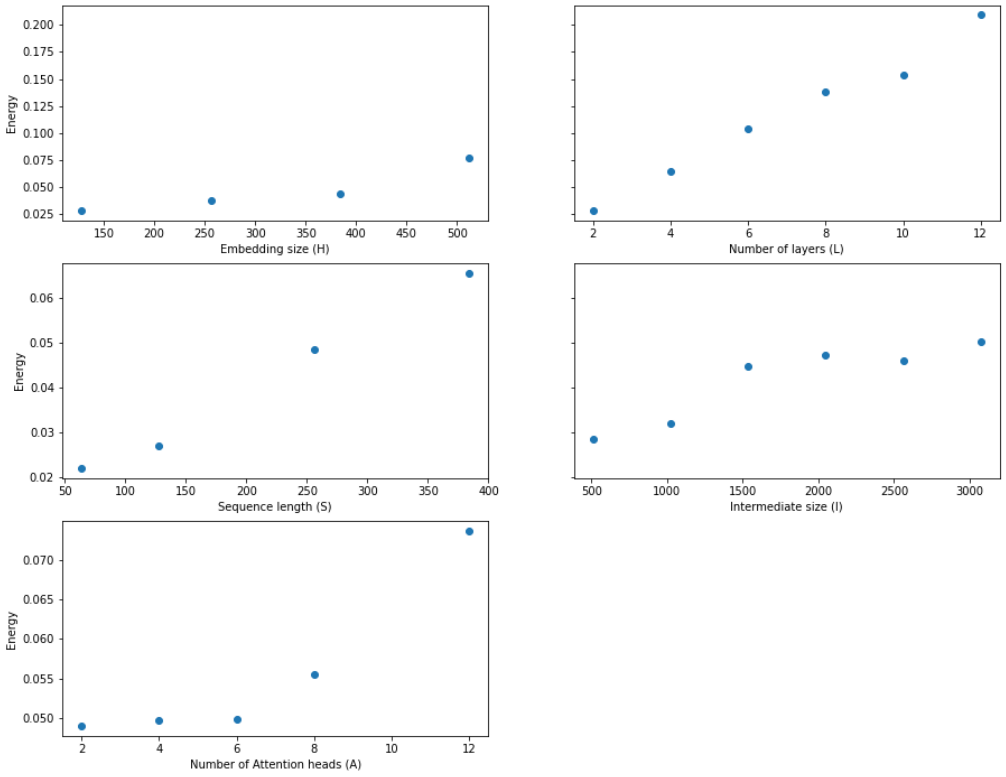

Figure 4.4: Energy consumption on GPU vs. BERT architectural parameters

Table 4.2: BERT design space on GPU

Architectural parameter	Values
H	128,256,384,512
S	64,128,256
L	2,4,6,8,10,12
A	2,4,8
I	512,1024,2048,3072

we reduce the selection on H, S, and I, and add more selections on A to avoid an explosion of the design space. The resulting design space consists of 864 models as presented in Table 4.2.

4.1.5 DistilBERT Architectural Parameter Selections

DistilBERT [16] is a lightweight transformer-based model used for embedded devices trained by applying knowledge distillation on BERT-base model. The general architecture of DistilBERT is the same as BERT with $H = 768, S = 512, L = 6, A = 12, I = 3072$. In addition to the BERT model design space, we add more data points around the architecture of DistilBERT. Table 4.3 presents the additional parameter selections.

Table 4.3: DistilBERT design space

Architectural parameter	Values
H	512,640,768, 896
S	512,1024
L	8,10,12,14,16
A	8
I	2560, 3072, 4096

4.1.6 MobileBERT Models

MobileBERT [2] has bottleneck structures in the encoder layers as shown in Figure 3.4 which adds 3 more parameters (embedding size, bottleneck size, number of FFNs in an embedding layer) in the design space. The bottleneck structure is a fully connected layer with fewer neurons than the previous layer. With the addition of architectural parameters, we reduce

Table 4.4: MobileBERT design space

Architectural parameter	Values
H	256,512
S	512,1024
L	12,16,20,24
A	$H/64$, $H/32$
I	$H \times 2$,$H \times 4$
E	64,128,192
B	64,128,192
F	2,4,6

each parameter's selection to reduce the number of data points in the design space. We follow the design of the BERT-base model of setting the number of attention heads and intermediate size based on the size of the embedding layer (e.g., $I = 2 * H$). With the constraints shown in 4.4, the overall design space consists of 432 models.

4.2 Energy Measurement Framework

Taking accurate energy measurements is vital and many factors such as temperature may cause variations in the measurement. Here we present our method of taking energy measurements for each candidate model inference on the target device. For each model, we profile the power consumption for a period of time that can be broken down into the following:

- **Idle time**: the system stays idle for a period of time, in our case, we select 30 seconds of idle time. The average power consumption for this period of time is the idle power

which we will use in the energy calculation.

- **Warm up time**: system power is temperature-dependent [70]. 60 seconds of warm-up inference is required to have stable measurements. We ran experiments that show the temperature of the board stays consistent with 60 seconds of warm-up.

- **Inference time**: multiple inferences for each model are done on the target board to get an accurate measurement. For each model, we continuously run approximately 120 seconds of inference. The actual inference time may vary as we run each inference cycle completely from start to finish. We calculate the average inference latency for this period and use it in the calculation of energy consumption.

- **Cool down time**: continuous inference on the target device may cause overheating which leads to increased latency which results in inaccurate energy measurement [77]. We set the system to idle for 30 seconds in between each measurement to maintain the temperature of the board.

We calculate the energy consumption based on the inference latency and the power consumption.

Energy consumption per inference:

$$E = \int_{t_0}^{t_1} (P_{active}(t) - P_{idle}(t))\, dt$$

where t_0 and t_1 represent the inference start and finish time, P_{active} represents the power consumption during inference and P_{idle} represents the power during system idle. Therefore, taking the difference between the two power consumption indicates the power consumed solely for inference.

4.3 Energy Prediction Framework

We use a DNN model to build an energy predictor for BERT inference on edge devices. Prior works [61, 64, 72] used linear regression models with Performance Monitoring Counters (PMCs) data to predict energy consumption. Using a linear regression model inherently assumes a linear relationship between input features and the output. However, this assumption is not proven in energy modelling [73]. Some research works use DNN [73, 74] with PMCs data as input features. Although these methods achieve higher accuracy in prediction, using PMCs is inefficient and sometimes infeasible due to the limitation of hardware architecture dependency, as the PMCs available for different hardware vary. Therefore, we choose DNN with architectural parameters as input features to predict energy consumption.

The DNN model we use consists of 4 fully connected layers and 1 dropout layer to prevent overfitting. Each layer consists of a different number of neurons listed as the following: {2048,1024,512,128}. For each design space, we divide the dataset into 80% train set and 20% test set. Within the train set, we set aside 20% of data as the validation set. Our model

is trained with 100 epochs and we keep the best set of parameters in history.

4.4 Fast Nondominated Sorting Genetic Algorithm

We use the fast nondominated sorting algorithm in NSGA-II [71] to find the POFs in a multi-objective design space. Searching for POFs in a large design space using an exhaustive approach is computationally expensive as the time complexity is $O(MN^3)$. Where M is the number of objectives and N is the number of models in the design space. Work in [71] provides a faster algorithm for finding POF with time complexity $O(MN^2)$. For each model p in the design space, two values are calculated: (1) the domination count n_p: which is the number of models that dominates p. (2) S_p: the set of models dominated by p. The first POF is the set models that have $n_p = 0$. To find the second POF, the following step is performed: for each model p in the first POF, visit each member in its S_p to reduce the n_p by 1. The resulting model with $n_p = 0$ forms the second POF.

In the evaluation of our energy predictor, we utilize this algorithm to find POFs in terms of accuracy and energy consumption. Finding different layers of POFs helps us gain a better understanding of the design space and better quantifies the ability of our energy predictor.

Chapter 5

Experiments and Results

In this chapter, we discuss the experimental setup for energy measurement and the evaluation of our energy predictors in terms of Mean Absolute Percentage Error (MAPE) metrics. We review experimental setups from previous works and discuss our energy measurement framework in details including the hardware platforms, the software framework and so on. We evaluate each energy predictor models in the design space of BERT models, DistilBERT models and MobileBERT models on the Hikey 970 ARM big.LITTLE CPU. In addition, we also evaluate the energy predictor model for BERT on the Mali GPU. We observe that DNN models perform the best in terms of MAPE across different design space and platforms.

5.1 Experimental setup

Obtaining accurate and stable power measurements is challenging since many factors contribute to noise in power (temperature, sampling frequency, etc.). We review existing power measurement setups from previous works on the Hikey 960 and Hikey 970 board [67, 78–81]. In the following, we break down the selection of measurement tools, CPU frequency, sampling rate, profiling execution, and energy measurement extraction. Based on these existing works, we established a working experimental setup for obtaining accurate and stable energy measurement.

5.1.1 Target Platform

We use the Hikey 970 board featuring a heterogeneous big.LITTLE system on chip (SoC), including four 2.36 GHz ARM Cortex-A73 high-performance cores and four 1.8 GHz ARM Cortex-A53 energy-efficient cores. Figure 5.1 shows an abstract block diagram of the CPU architecture of the Hikey 970. The Cortex-A73 core features a superscalar, variable length, and out-of-order, pipeline, 64 KB L1 instruction cache and 64 KB L1 data cache. The Cortex-A53 core features an in-order, pipeline, 32 KB L1 data cache, and 32 KB L1 instruction cache. The big cluster shares a 2MB L2 cache where the LITTLE cluster shares a 1MB L2 cache. Cache consistency within the two clusters is handled by the CCI-550, and interrupts are handled by the GIC-400 [46].

The Hikey 970 board also comes with a built-in Mali-G72 MP12 GPU configured with

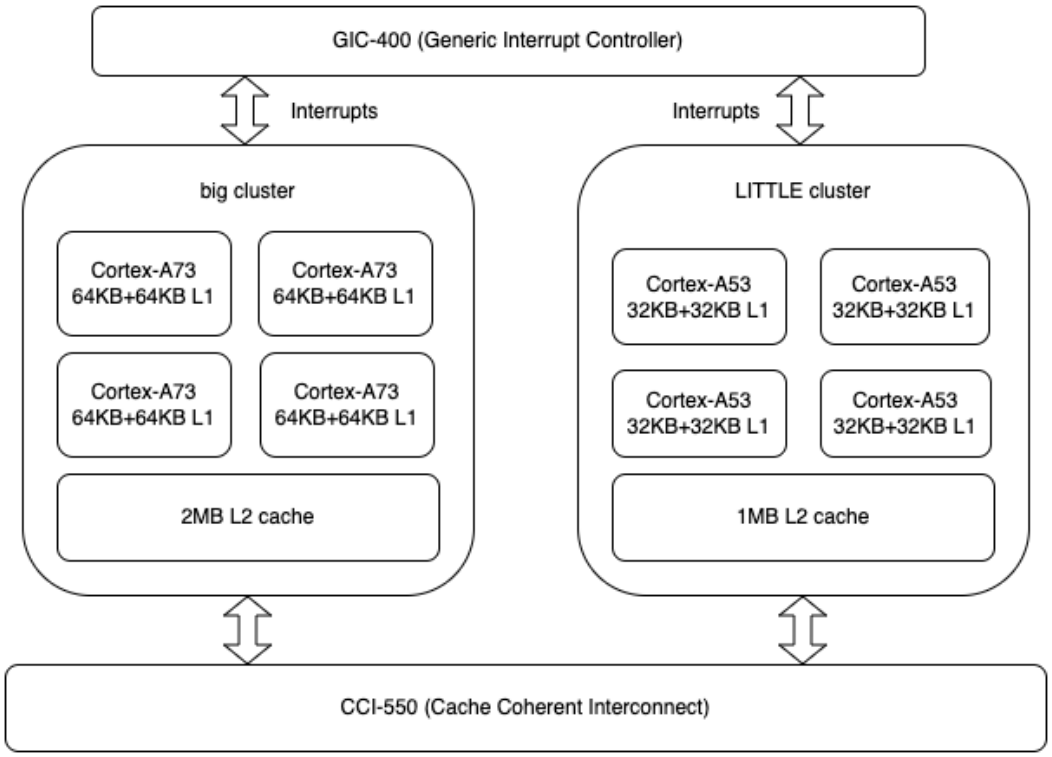

Figure 5.1: Hikey 970 CPU architecture abstract block diagram

two 512 KB L2 caches. Although dedicated accelerators such as GPU are proven to be more efficient, GPUs on mobile devices often lack programming support. On embedded devices, CPUs are the main platform for running Machine Learning algorithms [6]. In our experiment, we target both the CPU and the GPU on the Hikey 970 board.

5.1.2 Measurement Tools

Embedded system energy consumption is hard to measure as such system consist of multiple components. For some SoCs with ARM big.LITTLE architecture, power sensors on board

are able to capture energy consumption, however, this is not the case for the Hikey 960 and Hikey 970 boards. It is therefore impossible to distinguish the consumption of each component. Therefore, system-level energy consumption is measured using various tools.

Previous works using the Hikey 960 or Hikey 970 conduct energy measurements on various workloads. The type of workloads ranges from mobile applications such as Facebook to more computation-heavy machine learning algorithms, and computer vision tasks. Among those works, Monsoon Power Monitor is one of the most popular measurement tools used. Examples include: [67,78–81]. Other tools used include the National Instruments' DAQ unit X-series 646 [82], or the Keysight B2900A Series Precision Source/Measure Unit [83].

For our experiments, we wish to capture not only the energy consumption, but also CPU activities, hardware performance counters, and inference latency. Therefore we selected the ARM Streamline Performance Analyzer [84] to capture the performance profile of running an application on the Hikey 970 board. In support of capturing energy data, we use the ARM Energy probe. It collects the voltage, current and power of the target and is easily deployable with ARM Streamline. Figure 5.2 shows the energy measurement circuit. The operating voltage of the Hikey 970 is 12V and its shunt resister is $500m\Omega$. Figure 5.3 presents the overall setup. Energy consumption is calculated by taking the sum of instantaneous power samples for the duration of one inference. To ensure consistent and stable measurements, for each model, we run inference multiple times and take the average latency value to be used in the calculation.

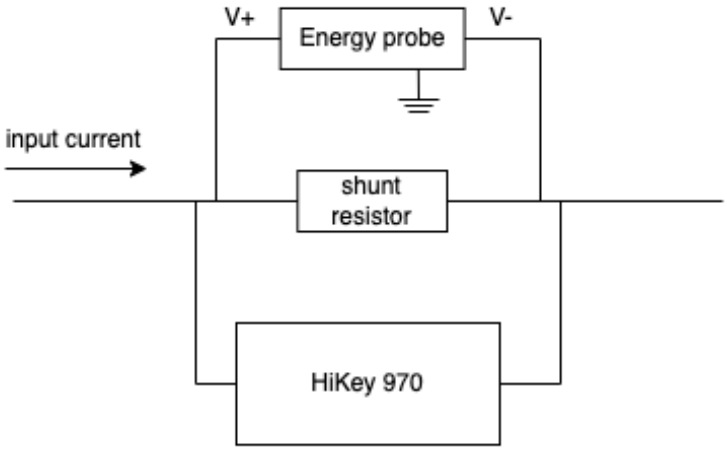

Figure 5.2: Energy measurement circuit diagram

5.1.3 CPU Frequency

The cores on the Hikey 960, Hikey 970 board can operate at a variety of voltage and frequency combinations. To avoid the impact of DVFS (Dynamic Voltage and Frequency Scaling) and ensure repeatable experiments, many research works set the CPU to run at the highest frequency [67]. In [82], they measure load time and energy consumption simultaneously on the Cortex-A9 processor running at 1.2GHz which is also the highest frequency [78]. We adopt the same strategy by setting both clusters to the highest frequencies. The big cluster of 4 ARM Cortex-A73 cores is set to 2.36GHz, whereas the little cluster of 4 ARM Cortex-A53 cores is set to 1.8GHz.

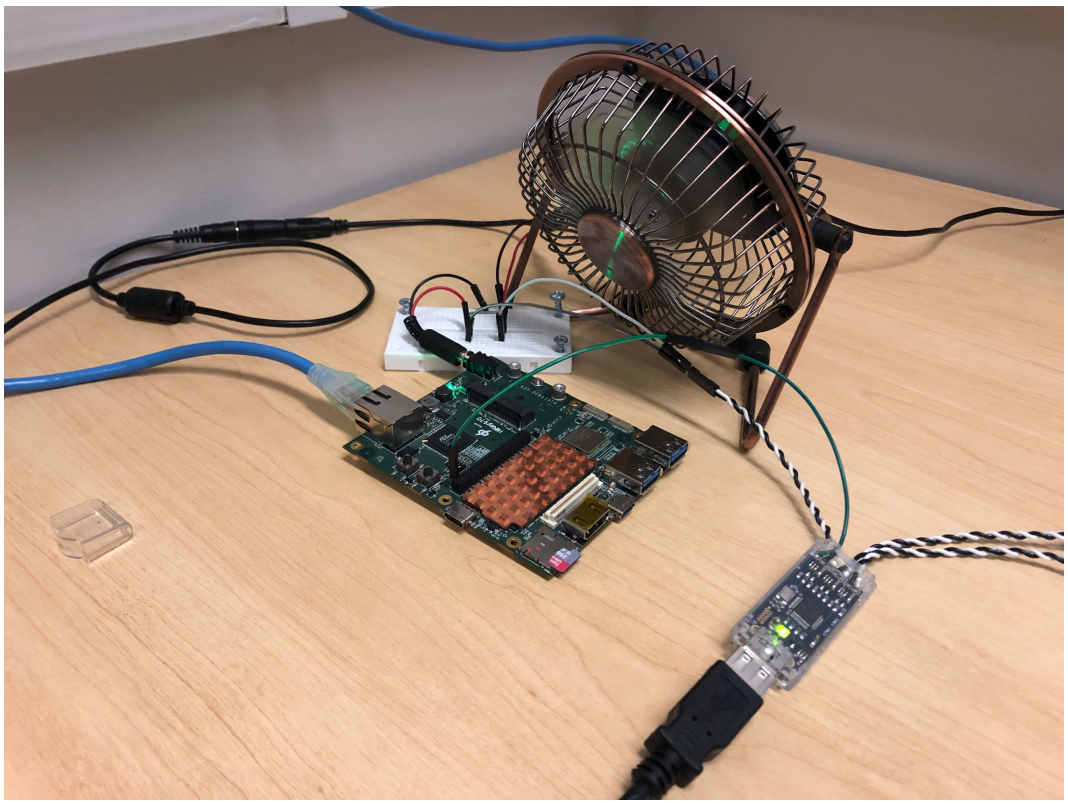

Figure 5.3: Energy measurement setup

5.1.4 Sampling Rate

In measurements of continuous signals, in this case, power, we can adjust the sampling rate so that the collected discrete measurements resemble the continuous signal. There's no clear way to determine the ground truth of energy consumption measurements. All measurements are approximations of the true energy consumption. In previous works, the sampling rate ranges from 3.48Hz to 5000Hz. In [68], the sampling rate is 3.48Hz. In [78], the authors record the current at 50Hz. In [80] all benchmark models run at least 10 times, the sampling frequency of the utilized power monitor is several magnitudes lower than the operating

frequency of the device hardware. The average of the energy data is recorded as the energy measurement. Their reported sampling rate is 5KHz. In [69], the sampling rate is also set to 5KHz.

In our experiments, the ARM Energy Probe has a sampling rate of 10KHz. It is significantly higher than other works. Since it's an external device attached to the target, it does not affect the overall performance of the target. However, target device activity is monitored and transmitted to the host device with the ARM Streamline Performance Analyzer. The data transmission process takes up CPU utilization and therefore affects the performance of inference. We set the sampling rate according to the ARM Streamline Performance Analyzer documentation to the standard value of 1KHz.

5.1.5 Thermal Control

The total power consumption of the CPU is the sum of dynamic power and static power. Dynamic power is further broken down into short-circuit power and switching power. Switching power is dissipated when charging or discharging internal capacitors. Short-circuit power is the power dissipated by an instantaneous short-circuit connection between the supply voltage and the ground at the time the logic gate switches state. Leakage power is the result of leakage current flowing through the transistors. The total CPU power:

$$P_{CPU} = P_{switching} + P_{short\_circuit} + P_{leakage} \tag{5.1}$$

The total CPU power is temperature-dependent. To ensure the repeatability of accurate power measurement, it is important to control the thermal effects. In other words, since power measurements can be different in different temperatures, it is important to keep the temperature at a constant value to produce constant and stable readings of power measurements.

The typically permissible operating temperature of a CPU is below 70 °C [70]. Cooling systems such as heat sinks, and heat pipes, are commercially used in modern CPUs. The Hikey 970 board does not come with a cooling system. We use a USB fan centers directly to the board. To ensure that the temperature is maintained at a constant value, we conduct experiments on the board. We run programs on the board which maximize the CPU activity while simultaneously sampling CPU temperature with a sample rate of 5Hz. Results show that with the USB fan, the temperature remains at around 40°C, whereas without the USB fan, the temperature increases from 50°C all the way up to 75 °C. Therefore, we keep the fan operating while collecting all the measurements.

5.1.6 Software Framework

With the increasing demand for deploying machine learning models on mobile devices, many frameworks are designed and optimized for on-device machine learning such as Tensorflow Lite [85] and Apache TVM [3]. We tested three frameworks for our experiments. The first we experimented with was Tensorflow Lite. Its python runtime package is just a fraction

of the size of the Tensorflow package offering the same ability to run Tensorflow models on mobile devices as Tensorflow package. Although the Tensorflow Lite package offers a simple installation process and supports the CPU platform, it does not support the Mali GPU on the Hikey 970 board.

Apache TVM is an open-source end-to-end machine learning compiler framework for CPUs and GPUs. It aims to enable machine learning on any hardware backend. The optimization stack is able to transform from any machine learning framework to machine code, providing portability across various backends. The overall pipeline is shown in Figure 5.4. Firstly, deep learning models from popular frameworks such as TensorFlow or PyTorch can be imported into the pipeline. The imported model is translated to Relay, a high-level language and intermediate representation (IR) for neural networks. Relay representations are then converted to lower tensor expressions. A search is then performed to find the best schedule based on cost models and on-device measurements. In the end, the lower-level representations are compiled down to machine code based on the target platform compiler. However, our experiments on GPU with TVM show that the performance on GPU is worse than on CPU big cores. This result shows that TVM does not guarantee performance on GPU. It also proves that software framework affects greatly the performance of running machine learning models.

We then experimented with ARM Compute Library on GPU and the results demonstrate faster inference than on CPU. Although ARM Compute Library shows better

performance, the framework requires much more effort for deployment compared to the other two frameworks. Different machine learning models require extensive refactoring to be run with ARM Compute Library. As an example, GELU operation is not provided in the ARM Compute Library. However, it is crucial for the software framework to provide better performance. We choose to use ARM Compute Library on GPU and Apache TVM on CPU.

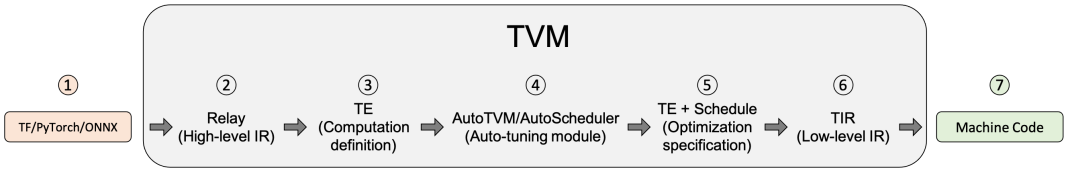

Figure 5.4: TVM model transformation to machine code [3]

5.2 Results

We evaluate our framework for energy prediction models in the design space of BERT models, DistilBERT models, and MobileBERT models. We compare various machine learning algorithms (polynomial regression, decision tree, AdaBoost, and DNN) for predicting energy consumption using the Mean Absolute Percentage Error (MAPE) metric and the percentage of data in a given error band. An error band specifies a boundary around the true value and it indicates the worst-case error.

Table 5.1: BERT on CPU big cluster: The performance of predictor in terms of MAPE, percentage of models that fit into 5%, 10%, and 20% error bands

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	0.4	11.1	27.0	45.0
Support Vector Machine	0.82	3.17	12.2	21.2
Decision Tree	0.27	34.9	63.5	77.8
AdaBoost	0.22	46.0	**75.7**	84.1
DNN	**0.15**	**57.7**	74.6	**85.7**

Table 5.2: BERT on CPU LITTLE cluster: The performance of predictor in terms of MAPE, percentage of models that fit into 5%, 10%, and 20% error bands

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	0.57	8.74	21.3	40.4
Support Vector Machine	0.87	2.19	7.65	18.0
Decision Tree	0.24	16.4	38.3	66.1
AdaBoost	0.19	26.2	43.2	73.2
DNN	**0.16**	**30.6**	**52.5**	**80.9**

5.2.1 Energy prediction on BERT

In the BERT design space, we split the collected data to a train set and a test set with a 0.8/0.2 ratio. For the DNN model, to prevent overfitting, 20% of data is sampled from the train set to form a validation set. In Table 5.1, we report the MAPE metric on the test

Table 5.3: BERT on GPU: The performance of predictor in terms of MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	1.84	6.38	14.9	20.2
Support Vector Machine	1.36	5.32	6.38	14.9
Decision Tree	0.50	18.0	31.2	51.1
AdaBoost	0.52	16.0	23.4	40.4
DNN	**0.16**	**67.0**	**76.6**	**83.0**

set for the CPU big cluster. We observe that the DNN model with a MAPE value of 0.15 performs better than the other regression models on the big cluster. We further examine the performance of each model by reporting the percentage of predicted energy in a given error-band. As an example, in Table 5.1 for error-band 20% and the DNN model, it means that 85.7% of the predicted energy value has less than 20% error from the true energy value. The results demonstrate that the DNN model achieves the best performance compared to the other regression models.

We perform experiments on the Mali GPU and the results are shown in Table 5.3. The DNN model performs significantly better than the other regression models with a MAPE of 0.16 and 83.0% data within the 20% error band, whereas the second best model has a MAPE of 0.16 and 51.1% data within the 20% error band.

5.2.2 Energy prediction on DistilBERT

Table 5.4: DistilBERT on CPU big cluster: The performance of predictor in terms of MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	**0.04**	**79.2**	91.7	**100**
Support Vector Machine	0.65	0	8.3	25.0
Decision Tree	0.07	29.2	70.8	**100**
AdaBoost	0.107	33.3	70.8	**100**
DNN	0.06	70.8	**95.8**	**100**

DistilBERT models have the same general architecture as BERT models with less number of layers. We use the energy predictor trained with the BERT design space and evaluate

Table 5.5: DistilBERT on CPU LITTLE cluster: The performance of predictor in terms of MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	0.11	25.0	54.2	83.3
Support Vector Machine	0.63	4.17	4.17	20.8
Decision Tree	0.13	16.7	37.5	83.3
AdaBoost	0.12	33.3	45.8	75.0
DNN	**0.06**	**50.0**	**75.0**	**95.8**

the predictor on the DistilBERT design space. Table 5.4 presents the performance of the energy predictor on CPU big cluster. The polynomial Regression model achieves the best performance in terms of MAPE. However, with the DNN model, 95.8% data falls into the 10% error band. On the CPU small cluster, the DNN model consistently performs the best in all metrics with 95.8% data in 20% error band. We observe that models perform generally better in this design space than in the BERT design space. This boost in performance is caused by the number of data used for this design space. We use the BERT design space data to train the energy predictor, whereas the predictor used in the previous section is trained with 80% of the BERT design space.

5.2.3 Energy prediction on MobileBERT

Energy predictors for MobileBERT models are trained by a train set of 80% data of the design space. The predictors are evaluated based on the rest 20% data. In Table 5.6 and Table 5.7, we present the energy prediction results in the MobileBERT design space. In terms of the MAPE metric, polynomial regression and DNN achieves the same performance

Table 5.6: MobileBERT on CPU big cluster: The performance of predictor in terms of MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	0.28	42.4	63.6	75.8
Support Vector Machine	1.8	6.06	6.06	9.09
Decision Tree	0.15	33.3	54.5	90.9
AdaBoost	0.13	45.5	69.7	87.9
DNN	**0.09**	**48.5**	**69.7**	**93.9**

Table 5.7: MobileBERT on CPU LITTLE cluster: The performance of predictor in terms of MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	**0.15**	**15.2**	27.3	66.7
Support Vector Machine	0.60	6.06	9.09	24.2
Decision Tree	0.18	24.2	36.4	54.5
AdaBoost	0.16	18.2	27.3	66.7
DNN	**0.15**	**15.2**	**33.3**	**69.7**

of 0.15. To further identify the better-performing model, we examine the percentage of test data in a given error band. We observe that DNN performs the best with 69.7% of test data in the 20% error-band. However, the predictor's performance on the LITTLE cluster is significantly lower than on the other design space. This drop in performance is due to the limited number of measurements collected. The BERT design space consists of 1080 models whereas the MobileBERT design space consists of only 432 models.

Chapter 6

Case Study on DynaBERT

We evaluate our BERT-base energy predictor model on DynaBERT [47] to examine its generalization capability in a new design space and its ability to find POFs in a multi-objective (accuracy and energy) problem. We select DynaBERT for two main reasons: (1) training and fine-tuning BERT models is computationally expensive. However, it is not the case for DynaBERT which is a kind of pre-trained supernet; (2) DynaBERT does not require fine-tuning for downstream tasks, and therefore, obtaining accuracy metrics is easier compared to other BERT-like models. Recall that BERT-base and DynaBERT supernet have the same architecture. However, the training set models does not include any sub-network models derived from DynaBERT. Therefore, training the energy predictor on BERT-base models and evaluating it on models derived from DynaBERT supernet would be feasible.

In the following, we present the design space derived from DynaBERT. We demonstrate

our energy predictor correctly predict 14 out of 17 models from the true POF.

6.1 Design Space

DynaBERT is a supernet with the same architecture as BERT-base. Sub-networks are generated from this supernet by varying the width and depth multipliers. If width and depth multipliers are set to one, the generated sub-network is BERT-base. The width parameter directly controls two architectural parameters of BERT model: the number of attention heads (A) and the intermediate size of FFN (I). The depth parameter controls the number of layers (L). Detailed explanations on BERT parameter architectures are presented in section 3.1. For a given width multiplier m_w, the sub-network retains the leftmost $\lfloor m_w A \rfloor$ attention heads and the leftmost $\lfloor m_w I \rfloor$ neurons in the FFN layer. Since the width parameter controls two architectural parameters, a given A corresponds to a fixed I. For a given depth multiplier $\lfloor m_d \rfloor$, the sub-network retains the first $\lfloor m_w L \rfloor$ encoder layers of the supernet.

In this regard, we establish our DynaBERT design space by varying the width and depth multipliers in DynaBERT from 0.05 to 1 with a step size of 0.05. This configuration results in 240 models and the values are presented in Table 6.1.

Table 6.1: DynaBERT design space

Architectural parameter	Values
H	768
S	512
L	1,2,3,4,5,6,7,8,9,10,11,12
A	1,2,3,4,5,6,7,8,9,10,11,12
I	153,307,460,614,768,921,1075,1228,1382,1536, 1689,1843,1996,2150,2304,2457,2611,2764,2918,3072

Table 6.2: The performance of predictors in terms of ADRS, MAPE, percentage of models in 5%, 10%, 20% error band

Regression Model	ADRS	MAPE	5% error band	10% error band	20% error band
Polynomial Regression	0.650	1.23	6.67	12.5	29.2
Support Vector Machine	0.499	0.47	12.5	23.75	45.0
Decision Tree	0.464	0.22	12.1	21.7	49.6
AdaBoost	0.464	0.20	10.0	24.2	50.4
DNN	**0.002**	**0.14**	**22.5**	**43.8**	**81.2**

6.2 Experiments and Results

Our DNN energy predictor trained with BERT base models achieves a MAPE of 0.14, which is the best compared to the other regression techniques as shown in Table 6.2. Among the 240 models from the DynaBERT design space, 81.2% models are predicted to have errors within 20% of the true energy value. In order to utilize our energy predictor model in hardware-aware NAS, we need to examine the energy predictor's ability to predict Pareto-optimal Front (POF) in the DynaBERT design space. In hardware-aware NAS, the problem is to find a set of models that are optimized for two objectives: performance and hardware

cost. This set of models forms a boundary in the multi-objective design space and is called Pareto-optimal solutions. The definition of POF is that there does not exist another solution in the design space that is more optimal in one objective and does not sacrifice in another objective [86]. We use Average Distance from Reference Set (ADRS) [87] to quantitatively measure the closeness from the predicted POF to the true POF. Average Distance from Reference Set (ADRS) is used to measure the distance between a reference Pareto-optimal Front (R) and an approximation set A [88]:

$$ADRS(R, A) = \frac{1}{|R|} \sum_{x \epsilon R} (\min_{a \epsilon A} \{d(x, a)\})$$

The benefit of using ADRS metric is that it not only considers the distance but also measures if the shape of the reference set is captured. If the ADRS between the true POF and the predicted POF is close to zero, it indicates that the predicted POF resembles the true POF.

We create the multi-objective design space with two objectives: accuracy and energy. We select the QNLI task from the GLUE benchmark to measure the accuracy metric. We use the fast nondominated sorting algorithm [71] with a detailed explanation in Section 4.4 to find the true POF and predicted POF. The fast nondominated sorting algorithm sorts layers of POFs in a design space, where the first layer is the POF of the original design space and the second layer POF is the POF of the design space excluding the design points in the first layer POF. We observe that although the error rate of the DNN predictor is high

(just 81.2% of design points lie in the 20% error band), the relative distance between the predicted POF set based on the DNN predictor and the real POF set is the lowest (0.002) in terms of ADRS. In Figure 6.1, we plot the true POF and predicted POF. Note that for the predicted POF, first, we use the predicted energy values to find POF, then, we substitute the predicted energy values in the found POF set with the true energy consumption derived from measurements (that is why in the bottom of the predicted POF, red line, it seems that there is a non-dominated optimal solution). The true POF consists of 17 candidate models. 14 of those are in the predicted POF, while the other 3 are found in the second layer of the predicted POF. These results show that our energy prediction framework is able to correctly find POF in a large design space.

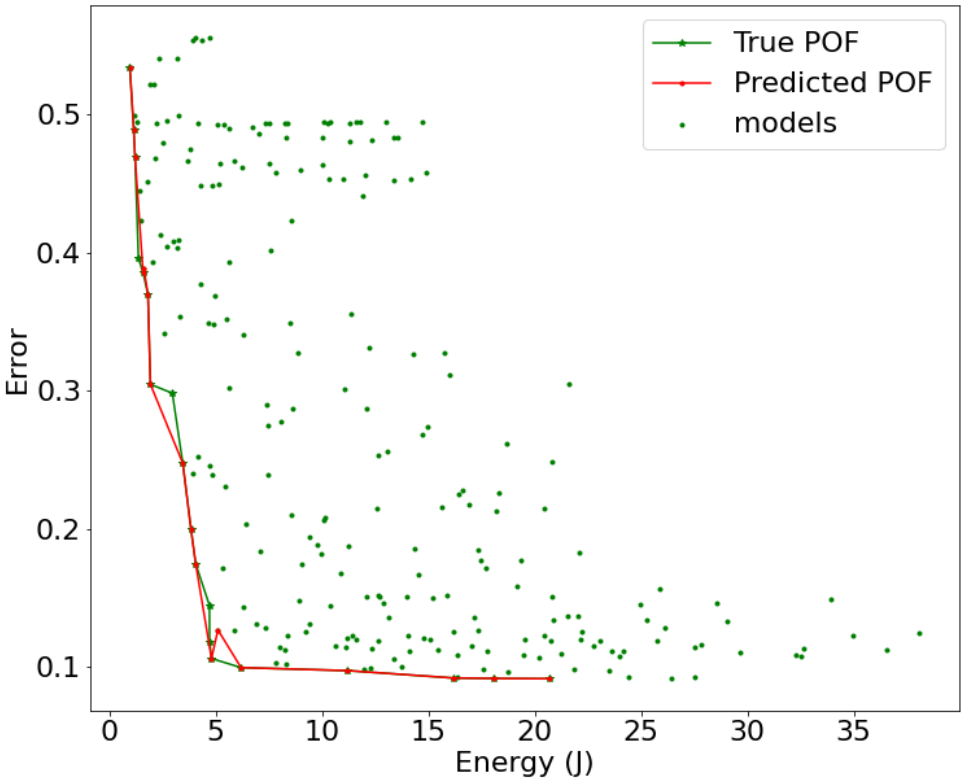

Figure 6.1: True and predicted Pareto-optimal fronts of DynaBERT design space in a 2D design space of error (1-accuracy) and energy.

Chapter 7

Conclusion and Future Work

In this thesis, we introduced a BERT inference energy consumption modeling methodology on ARM big.LITTLE architecture and Mali GPU. We established an energy measurement framework for taking accurate and stable system energy measurements. Measurement data of a candidate model can be obtained in 240 seconds with our energy measurement framework. We studied the relationships between BERT architectural parameters and inference energy consumption. We found that each architectural parameter has different effects on inference energy consumption. We also found that the effects of architectural parameters are different on different hardware platforms. The number of attention heads (A) contributes to less than 10% change in energy on CPU small clusters and big clusters, however, this phenomenon is not shown on the GPU, where increasing the number of attention heads (A) results in a larger impact on energy consumption.

Based on our observations, we built representative design spaces for each model to reduce the time for energy measurement. The design space of BERT model on the CPU consists of 1080 candidate models.

For each design space, we used 80% of the measured data as training data and trained 5 regression models (polynomial regression, support vector machine, DNN, etc.). Results show that DNN models perform consistently better than other regression techniques since DNNs are better at capturing non-linear relationships. The energy predictor of BERT models on CPU big cluster has a MAPE of 0.15 and 85.7% of the test data fall in the 20% error-band.

We also performed a case study on DynaBERT to evaluate our predictor's ability to predict POFs in a multi-objective problem and the ability to generalize to a new design space. In the experiment, we used the energy predictor trained with BERT base models. The DynaBERT design space consists of 240 different models that none of the models are present in the training set. We compared the true POF and predicted POF in terms of model accuracy and energy consumption. Our predictor correctly identifies 82.4% of the models in the true POF. In addition, 100% of the models in the true POF are identified within the first and second layers of the predicted POF sets.

In this thesis, we demonstrated that our predictors which are trained with relatively few on-device measurements perform well in a multi-objective problem and generalize to new design spaces. Future work to improve upon our work includes:

1. We run model inference in a homogeneous computing setting. An improvement would

be running models in a heterogeneous setting that utilizes CPU small cores, CPU big cores and GPU for example.

2. Our methodology can be applied to efficient hardware-aware NAS. However, it is computationally expensive and time-consuming to measure the accuracy of BERT models which is required for NAS. It will be interesting to study the possibility of building accuracy predictor models that serves the same purpose as our energy predictor.

Bibliography

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[2] Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou, "MobileBERT: a compact task-agnostic BERT for resource-limited devices," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, (Online), pp. 2158–2170, Association for Computational Linguistics, July 2020.

[3] T. Chen, T. Moreau, Z. Jiang, H. Shen, E. Q. Yan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "Tvm: end-to-end optimization stack for deep learning," *arXiv preprint arXiv:1802.04799*, vol. 11, p. 20, 2018.

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] M. Mohammadi, A. Al-Fuqaha, S. Sorour, and M. Guizani, "Deep learning for iot big data and streaming analytics: A survey," *IEEE Communications Surveys Tutorials*, vol. 20, no. 4, pp. 2923–2960, 2018.

[6] M. Nadeski, "Bringing machine learning to embedded systems," *Texas Instruments*, pp. 1–5, 2019.

[7] Z. Zhang and A. Z. Kouzani, "Implementation of dnns on iot devices," *Neural Computing and Applications*, vol. 32, no. 5, pp. 1327–1356, 2020.

[8] S. Branco, A. G. Ferreira, and J. Cabral, "Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey," *Electronics*, vol. 8, no. 11, p. 1289, 2019.

[9] H. Cai, C. Gan, T. Wang, Z. Zhang, and S. Han, "Once-for-all: Train one network and specialize it for efficient deployment," *arXiv preprint arXiv:1908.09791*, 2019.

[10] A. Fan, E. Grave, and A. Joulin, "Reducing transformer depth on demand with structured dropout," *arXiv preprint arXiv:1909.11556*, 2019.

[11] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?," *Advances in neural information processing systems*, vol. 32, 2019.

[12] M. A. Gordon, K. Duh, and N. Andrews, "Compressing bert: Studying the effects of weight pruning on transfer learning," *arXiv preprint arXiv:2002.08307*, 2020.

[13] A. Raganato, Y. Scherrer, and J. Tiedemann, "Fixed encoder self-attention patterns in transformer-based machine translation," *arXiv preprint arXiv:2002.10260*, 2020.

[14] V. Sanh, T. Wolf, and A. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," *Advances in Neural Information Processing Systems*, vol. 33, pp. 20378–20389, 2020.

[15] S. Sun, Y. Cheng, Z. Gan, and J. Liu, "Patient knowledge distillation for bert model compression," *arXiv preprint arXiv:1908.09355*, 2019.

[16] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[17] X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu, "Tinybert: Distilling bert for natural language understanding," *arXiv preprint arXiv:1909.10351*, 2019.

[18] I. Turc, M.-W. Chang, K. Lee, and K. Toutanova, "Well-read students learn better: On the importance of pre-training compact models," *arXiv preprint arXiv:1908.08962*, 2019.

[19] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou, "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers," *Advances in Neural Information Processing Systems*, vol. 33, pp. 5776–5788, 2020.

[20] C. Xu, W. Zhou, T. Ge, F. Wei, and M. Zhou, "Bert-of-theseus: Compressing bert by progressive module replacing," *arXiv preprint arXiv:2002.02925*, 2020.

[21] A. Bhandare, V. Sripathi, D. Karkada, V. Menon, S. Choi, K. Datta, and V. Saletore, "Efficient 8-bit quantization of transformer neural machine language translation model," *arXiv preprint arXiv:1906.00532*, 2019.

[22] O. Zafrir, G. Boudoukh, P. Izsak, and M. Wasserblat, "Q8bert: Quantized 8bit bert," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pp. 36–39, IEEE, 2019.

[23] S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer, "Q-bert: Hessian based ultra low precision quantization of bert," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 8815–8821, 2020.

[24] A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin, "Training with quantization noise for extreme model compression," *arXiv preprint arXiv:2004.07320*, 2020.

[25] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, "Gobo: Quantizing attention-based nlp models for low latency and energy efficient inference," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 811–824, IEEE, 2020.

[26] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[27] F. N. Iandola, A. E. Shaw, R. Krishna, and K. W. Keutzer, "Squeezebert: What can computer vision teach nlp about efficient neural networks?," *arXiv preprint arXiv:2006.11316*, 2020.

[28] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," *arXiv preprint arXiv:1611.01578*, 2016.

[29] D. Chen, Y. Li, M. Qiu, Z. Wang, B. Li, B. Ding, H. Deng, J. Huang, W. Lin, and J. Zhou, "Adabert: Task-adaptive bert compression with differentiable neural architecture search," *arXiv preprint arXiv:2001.04246*, 2020.

[30] S. C. Smithson, G. Yang, W. J. Gross, and B. H. Meyer, "Neural networks designing neural networks: multi-objective hyper-parameter optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2016.

[31] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1586–1595, 2018.

[32] H. Wang, Z. Wu, Z. Liu, H. Cai, L. Zhu, C. Gan, and S. Han, "Hat: Hardware-aware transformers for efficient natural language processing," *arXiv preprint arXiv:2005.14187*, 2020.

[33] T.-J. Yang, Y.-H. Chen, J. Emer, and V. Sze, "A method to estimate the energy consumption of deep neural networks," in *2017 51st asilomar conference on signals, systems, and computers*, pp. 1916–1920, IEEE, 2017.

[34] M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le, "Mnasnet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

[35] C.-H. Hsu, S.-H. Chang, J.-H. Liang, H.-P. Chou, C.-H. Liu, S.-C. Chang, J.-Y. Pan, Y.-T. Chen, W. Wei, and D.-C. Juan, "Monas: Multi-objective neural architecture search using reinforcement learning," *arXiv preprint arXiv:1806.10332*, 2018.

[36] T.-J. Yang, A. Howard, B. Chen, X. Zhang, A. Go, M. Sandler, V. Sze, and H. Adam, "Netadapt: Platform-aware neural network adaptation for mobile applications," in

Proceedings of the European Conference on Computer Vision (ECCV), pp. 285–300, 2018.

[37] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–7, IEEE, 2019.

[38] M. Abdelgawad, S. H. Mozafari, J. J. Clark, B. H. Meyer, and W. J. Gross, "Bertperf: Inference latency predictor for bert onarm big.little multi-core processors," *Accepted in 2022 IEEE Workshop on Signal Processing Systems (SiPS)*, 2022.

[39] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," *arXiv preprint arXiv:1812.00332*, 2018.

[40] A. Marchisio, A. Massa, V. Mrazek, B. Bussolino, M. Martina, and M. Shafique, "Nascaps: A framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks," in *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1–9, IEEE, 2020.

[41] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer, "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10734–10742, 2019.

[42] M. Berman, L. Pishchulin, N. Xu, M. B. Blaschko, and G. Medioni, "Aows: Adaptive and optimal network width search with latency constraints," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11217–11226, 2020.

[43] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. Yang, "Runtime performance prediction for deep learning models with graph neural network," tech. rep., Technical Report MSR-TR-2021-3. Microsoft, 2021.

[44] H. Lee, S. Lee, S. Chong, and S. J. Hwang, "Hardware-adaptive efficient latency prediction for nas via meta-learning," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27016–27028, 2021.

[45] H. Tsai, J. Ooi, C.-S. Ferng, H. W. Chung, and J. Riesa, "Finding fast transformers: One-shot neural architecture search by component composition," *arXiv preprint arXiv:2008.06808*, 2020.

[46] "Hikey 970." `https://www.hisilicon.com/en/products/Kirin/Kirin-flagship-chips/Kirin-970`, 2017.

[47] L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu, "Dynabert: Dynamic bert with adaptive width and depth," *Advances in Neural Information Processing Systems*, vol. 33, pp. 9782–9793, 2020.

[48] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[49] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[50] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[51] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[52] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8612–8620, 2019.

[53] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[54] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "Mcpat: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *Proceedings of the 42nd annual ieee/acm international symposium on microarchitecture*, pp. 469–480, 2009.

[55] S. K. Rethinagiri, O. Palomar, R. Ben Atitallah, S. Niar, O. Unsal, and A. C. Kestelman, "System-level power estimation tool for embedded processor based platforms," in *Proceedings of the 6th Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, pp. 1–8, 2014.

[56] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[57] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5687–5695, 2017.

[58] C. Yoon, S. Lee, Y. Choi, R. Ha, and H. Cha, "Accurate power modeling of modern mobile application processors," *Journal of Systems Architecture*, vol. 81, pp. 17–31, 2017.

[59] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh, "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural

network," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pp. 764–775, IEEE, 2018.

[60] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin, "Power-performance modeling on asymmetric multi-cores," in *2013 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, pp. 1–10, IEEE, 2013.

[61] M. J. Walker, S. Diestelhorst, A. Hansson, A. K. Das, S. Yang, B. M. Al-Hashimi, and G. V. Merrett, "Accurate and stable run-time power modeling for mobile and embedded cpus," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 1, pp. 106–119, 2016.

[62] Y. Zhang, Y. Liu, X. Liu, and Q. Li, "Enabling accurate and efficient modeling-based cpu power estimation for smartphones," in *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, IEEE, 2017.

[63] E. Cai, D.-C. Juan, D. Stamoulis, and D. Marculescu, "Neuralpower: Predict and deploy energy-efficient convolutional neural networks," in *Asian Conference on Machine Learning*, pp. 622–637, PMLR, 2017.

[64] C. F. Rodrigues, G. Riley, and M. Luján, "Synergy: An energy measurement and prediction framework for convolutional neural networks on jetson tx1," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and*

Applications (PDPTA), pp. 375–382, The Steering Committee of The World Congress in Computer Science, Computer …, 2018.

[65] H. Nagasaka, N. Maruyama, A. Nukada, T. Endo, and S. Matsuoka, "Statistical power modeling of gpu kernels using performance counters," in *International conference on green computing*, pp. 115–122, IEEE, 2010.

[66] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[67] J. Zhang, D. Zhang, X. Xu, F. Jia, Y. Liu, X. Liu, J. Ren, and Y. Zhang, "Mobipose: Real-time multi-person pose estimation on mobile devices," in *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*, pp. 136–149, 2020.

[68] E. Vasilakis, I. Sourdis, V. Papaefstathiou, A. Psathakis, and M. G. Katevenis, "Modeling energy-performance tradeoffs in arm big. little architectures," in *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, pp. 1–8, IEEE, 2017.

[69] M. Han, J. Hyun, S. Park, J. Park, and W. Baek, "Mosaic: Heterogeneity-, communication-, and constraint-aware model slicing and execution for accurate and efficient inference," in *2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 165–177, IEEE, 2019.

[70] J. Choi, M. Jeong, J. Yoo, and M. Seo, "A new cpu cooler design based on an active cooling heatsink combined with heat pipes," *Applied thermal engineering*, vol. 44, pp. 50–56, 2012.

[71] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[72] X. Ma, M. Dong, L. Zhong, and Z. Deng, "Statistical power consumption analysis and modeling for gpu-based computing," in *Proceeding of ACM SOSP Workshop on Power Aware Computing and Systems (HotPower)*, vol. 1, 2009.

[73] S. Song, C. Su, B. Rountree, and K. W. Cameron, "A simplified and accurate model of power-performance efficiency on emergent gpu architectures," in *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, pp. 673–686, IEEE, 2013.

[74] H. Wang and Y. Cao, "Predicting power consumption of gpus with fuzzy wavelet neural networks," *Parallel Computing*, vol. 44, pp. 18–36, 2015.

[75] T. Tambe, C. Hooper, L. Pentecost, T. Jia, E.-Y. Yang, M. Donato, V. Sanh, P. Whatmough, A. M. Rush, D. Brooks, *et al.*, "Edgebert: Sentence-level energy optimizations for latency-aware multi-task nlp inference," in *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 830–844, 2021.

[76] "Armdeveloper. 2016." `https://developer.arm.com/documentation/100614/0314/`. Accessed: 2022-10-04.

[77] T. Benoit-Cattin, D. Velasco-Montero, and J. Fernández-Berni, "Impact of thermal throttling on long-term visual inference in a cpu-based edge device," *Electronics*, vol. 9, no. 12, p. 2106, 2020.

[78] F. Mo, H. Haddadi, K. Katevas, E. Marin, D. Perino, and N. Kourtellis, "Ppfl: privacy-preserving federated learning with trusted execution environments," *arXiv preprint arXiv:2104.14380*, 2021.

[79] C. Ji, L.-P. Chang, R. Pan, C. Wu, C. Gao, L. Shi, T.-W. Kuo, and C. J. Xue, "Pattern-guided file compression with user-experience enhancement for log-structured file system on mobile devices," in *19th {USENIX} Conference on File and Storage Technologies ({FAST} 21)*, pp. 127–140, 2021.

[80] A. Schuler and G. Anderst-Kotsis, "Characterizing energy consumption of third-party api libraries using api utilization profiles," in *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 1–11, 2020.

[81] R. Liu, L. Garcia, Z. Liu, B. Ou, and M. Srivastava, "Secdeep: Secure and performant on-device deep learning inference framework for mobile and iot devices," in *Proceedings*

of the International Conference on Internet-of-Things Design and Implementation, pp. 67–79, 2021.

[82] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 13–24, IEEE, 2013.

[83] S. Wang, G. Ananthanarayanan, Y. Zeng, N. Goel, A. Pathania, and T. Mitra, "High-throughput cnn inference on embedded arm big. little multicore processors," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 10, pp. 2254–2267, 2019.

[84] "Armdeveloper. 2019." `https://developer.arm.com/documentation/101816/0802/`. Accessed: 2022-10-04.

[85] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[86] K. Miettinen, *Nonlinear multiobjective optimization*, vol. 12. Springer Science & Business Media, 2012.

[87] P. Czyżżak and A. Jaszkiewicz, "Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization," *Journal of multi-criteria decision analysis*, vol. 7, no. 1, pp. 34–47, 1998.

[88] G. Palermo, C. Silvano, and V. Zaccaria, "An efficient design space exploration methodology for multiprocessor soc architectures based on response surface methods," pp. 150 – 157, 08 2008.