IEEE Transactions on Artificial Intelligence, Volume 5, Issue 1, January 2024, pp. 192-204
DOI: 10.1109/TAI.2022.3229280

A Transistor Operations Model for Deep Learning Energy Consumption Scaling Law

Chen Li, Antonios Tsourdos and Weisi Guo, *Senior Member, IEEE*

***Abstract*—Deep Neural Networks (DNN) has transformed the automation of a wide range of industries and finds increasing ubiquity in society. The high complexity of DNN models and its widespread adoption has led to global energy consumption doubling every 3-4 months. Current energy consumption measures largely monitor system wide consumption or make linear assumptions of DNN models. The former approach captures other unrelated energy consumption anomalies, whilst the latter does not accurately reflect nonlinear computations. In this paper, we are the first to develop a bottom-up Transistor Operations (TOs) approach to expose the role of non-linear activation functions and neural network structure. As there will be inevitable energy measurement errors at the core level, we statistically model the energy scaling laws as opposed to absolute consumption values. We offer models for both feedforward DNNs and convolution neural networks (CNNs) on a variety of data sets and hardware configurations - achieving a 93.6% - 99.5% precision. This outperforms existing FLOPs-based methods and our TOs method can be further extended to other DNN models.**

***Impact Statement*—Deep learning is one of the fastest growth areas for computational resources (300,000x from 2012 to 2018, doubling every 3-4 months). Data centres are predicted to dominate over 20% of global energy consumption by 2030. Our proposed TOs model provides developers with a theoretical model to expose the important role of both (1) nonlinear activation functions and (2) DNN model structure in its energy consumption. This enables developers to trade-off between model performance and sustainability with 93.6% - 99.5% precision. Due to the consideration of both linear and non-linear operation in TOs, it can to some extent replace FLOPs/MACs count as a more accurate metric of DNN model complexity.**

***Index Terms*—Energy Consumption; Deep Learning; Model Architecture; Transistor Operations;**

I. Introduction

RAPIDLY increased Artificial Intelligence (AI) demand has generated a huge increase in computational resource requirement (300,000x from 2012 to 2018) [1]. Energy consumption in data centres around the world to maintain data and learn models will account for 10% of global energy consumption in 2025 and 20.9% in 2030 [2]. Whilst previous attempts in green communications have reduced networking energy consumption [3], Internet-of-Everyting will connect intelligence and it is important to reduce energy consumption across connectivity and autonomy [4]. The endless chasing of higher-precision in DNN spawns ultra-large-scale models, especially in computer vision (CV), natural language processing (NLP) [5], [6], and also communication networks [7] (see - Fig. 1*a*: YOLOR-D6 for CV with 174.7 million parameters in May 2021 [8]; openAI GPT-3 for NLP with 175 billion parameters in May 2020 [9]). Researchers in [10] argue that this trend is unfriendly to computational resources, energy and the global environment. Developers should carefully analyze the requirements (e.g., precision, robustness) and backgrounds (e.g., computational hardware, energy supply) of DNN tasks to make a trade-off between the performance and economy of DNN models. The network size of large-scale DNN models also barriers their deployment in energy-sensitive devices (e.g., Drones and remote sensors). Developers urgently need a theoretical method to analyze the scaling law of DNN model energy consumption during model configuration changes to design and select energy-efficient DNN models.

TABLE I
Definition of Abbreviations and Frequently Used Concepts

AF	Activation Function	IC	Integrated Circuit
ALU	Arithmetic Logic Units	LNL	Linear and Non-Linear
BO	Basic Operation	MAC	Multiply-Accumulate
BP	BackPropagation	ML	Machine Learning
CPU	Central Processing Unit	NLP	Natural Language Processing
CV	Computer Vision	PMC	Performance Monitoring Counters
DNN	Deep Neural Network	PR	Polynomial Regression
(D)RAM	(Dynamic) Random-Access Memory	RMSE	Root Mean Square Error
FLOPs	FLoating-point Operations	$\hat{R}^2$	Adjusted R-squared
FPGA	Field-Programmable Gate Array	SIMD	Single Instruction Multiple Data
FP	Floating-Point number	TO	Transistor Operation
FPU	Floating-Point Unit	TPU	Tensor Processing Unit
GELU	Gaussian Error Linear Unit	XOR	eXclusive OR
GPU	Graphical Processing Unit		

TOs: a theoretical metric of DNN complexity and calculation tasks, means how many transistor operations are involved in calculations
TOs method: the method to analyze DNN energy scaling with TOs
TOs model: the model we use to calculate DNN model TOs
Energy-efficiency: model performance over certain energy consumption. (higher - better performance within certain energy consumption)
Energy consumption: real energy consumption in Joule
Energy metric: theoretical metrics to represent DNN calculation tasks, these metrics related to DNN real energy consumption in Joule

Indeed, there are now widespread efforts to reduce the size of neural network architectures both post-training [11], federation to the edge [12], and more recently during training [13]. Current theoretical DNN complexity metric FLOPs [14] and MACs [15] only consider linear operations (e.g., multiplications and additions) without non-linear operations (e.g., root operation in activation functions). However, non-linear operations are a non-negligible part of some DNN models [16]

Chen Li, Antonios Tsourdos and Weisi Guo are with Digital Aviation Research Technology Centre (DARTeC), Cranfield University, Bedford, United Kingdom. Weisi Guo is also with the Alan Turing Institute, London, United Kingdom. Corresponding author: weisi.guo@cranfield.ac.uk. We acknowledge EC H2020 DAWN4IoE - Data Aware Wireless Network for Internet-of-Everything (778305) for supporting this work. Data is open available and details are in supplementary materials.

© 2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collective works, for resale or redistribution to servers or lists, or reuse of any copyrighted component of this work in other works

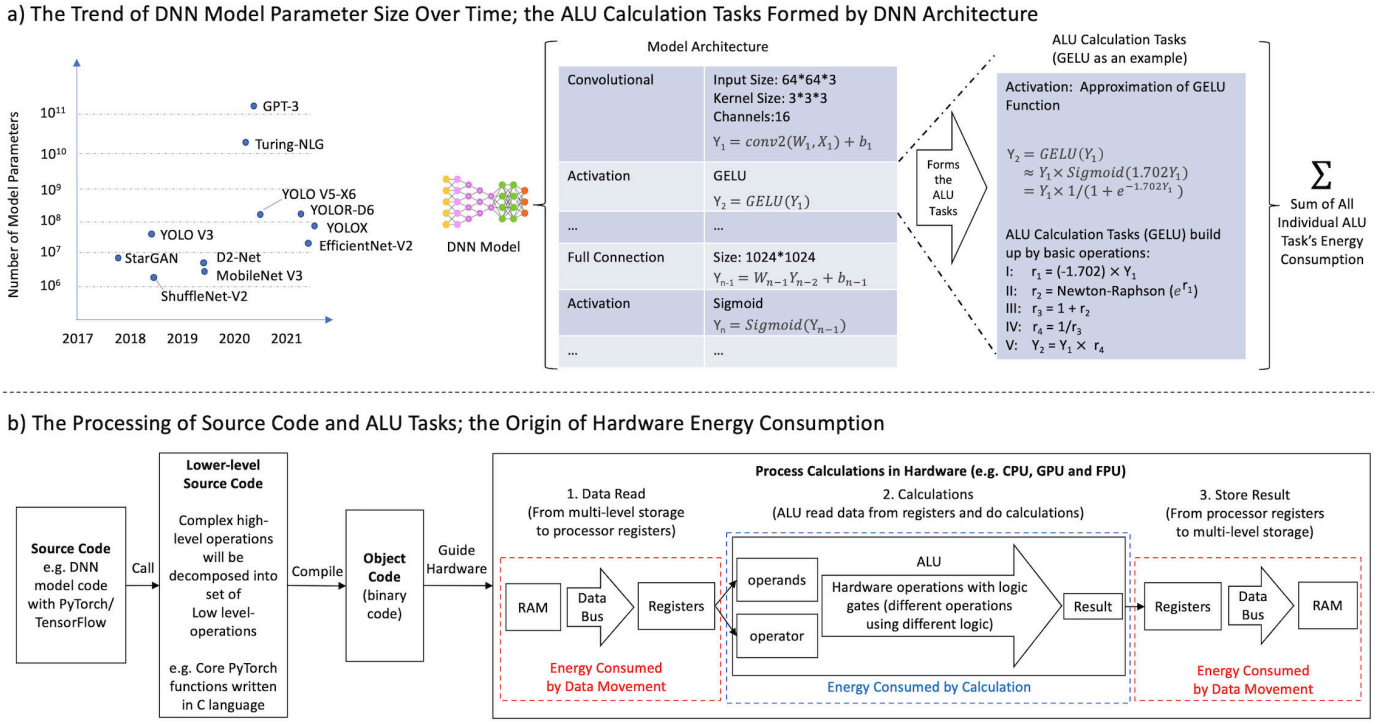


Fig. 1. The Trend of DNN Model Parameter Size Over Time and the Origin of DNN Energy Consumption

that the amount of calculation tasks and energy cost for doing a non-linear operation is always higher than linear ones [17], [18]. Thus, using FLOPs/MACs in analysing the scaling of DNN model complexity and energy consumption is not precise enough. Simultaneously, binary neural network [19], [20], low precision arithmetic [21], low precision number [22]–[24] and high-efficiency operators [25]–[27] they affect the DNN energy consumption, without effect in model FLOPs/MACs. To solve the issue above, we proposed a more accurate TOs method to analyze DNN model calculation tasks and energy consumption, which considers linear, non-linear operations and floating-point format. We show the use of TOs as a metric of DNN complexity could boost the precision of DNN energy estimation models. Details are introduced later, frequently used abbreviations are listed in Table I.

A. Review on Deep Neural Network Energy Model

DNN energy consumption in training, validation, and testing can all be related to the model complexity, data size, and hardware implementation [28]. As shown in Fig. 1*a*, the DNN model architecture (incl. the activation function) determines the resulting execution order of the equivalent arithmetic logic units (ALU) tasks. Within this, the input data affects the DNN model hyper-parameters which is part of the ALU tasks and overall they all contribute to different energy consumption.

Fig. 1*b* (left to right) briefly demonstrates the running of DNN model code in hardware and resulting energy consumption. Once the DNN model source code (e.g. Python) is run, learning frameworks (e.g., PyTorch, TensorFlow) will call the relevant core functions written in lower-level languages (e.g., C/C++). For example, the Gaussian Error Linear Unit (GELU) [29] operation will be decomposed into a set of ALU-supported instructions (see - Fig. 1*a*). Lower-level source codes will be further compiled into object codes to guide data reading, calculation and result storage [30]. The calculation in ALU follows a designed process [31]. Firstly, operands will be called from data storage (DRAM etc.) to the processor registers through data bus. The energy for moving one unit data varies with the memory hierarchy. Secondly, do the specific hardware operation (specified in operator) on operands, and generate calculation energy consumption. Thirdly, the result will be temporarily stored in registers and then moved to other levels of storage. As explained above, mainly energy consumption of code running is for data movement and calculation. We provide a review of different processors, please refer to more details in the supplementary material file.

Based on the information above, any comprehensive analysis of DNN energy model must encompass:

- Analyze the origin of hardware energy consumption at transistor operation (TO) and data movement level
- Translate linear and non-linear DNN activation functions into the real calculation tasks executed by ALU
- Propose a TO based energy metric that is generalised across diverse hardware operations
- Develop a regression model to quantify the energy scaling with model configuration

B. Deep Learning's Energy Breakdown

Current literature shows that DNN models primarily consume two types of energy: (1) calculation energy (as described earlier), and (2) data movement and memory access energy [6]. The latter constitutes a significant part of the energy

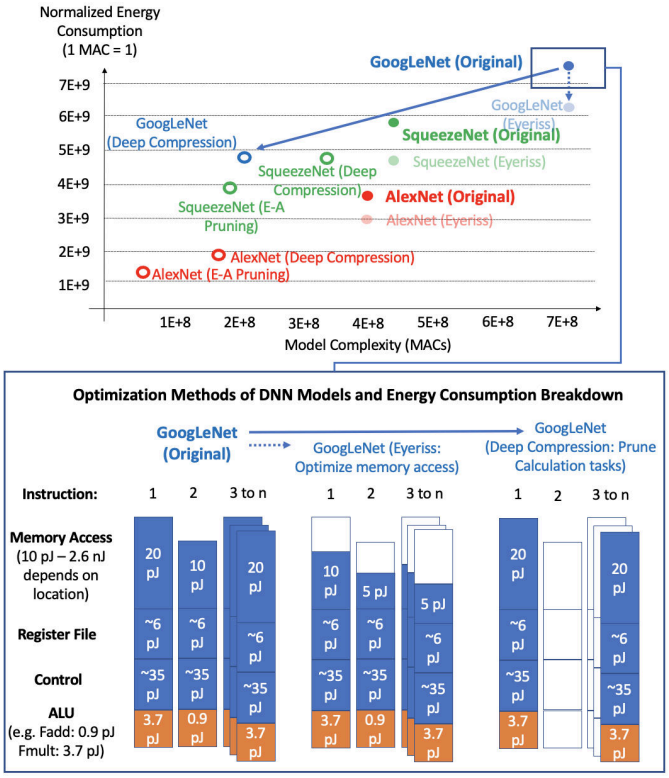


Fig. 2. (top) Normalized Energy Consumption and (bottom) Energy Breakdown of DNN Models. Data from [17], [34])

consumption, especially for large data sets [32]–[34]. In the execution of each instruction, data calling from hierarchy storage dominate up to *90%* energy consumption while calculations in ALU count less than *30%* [35].

As shown in Fig. 2(bottom), the data to be moved depends on operation tasks in instructions. Data movement optimization methods like Eyeriss [35] and MONeT [36] significantly reduce memory access energy consumption with higher data multiplexing rate and low-cost storage usage. Network pruning and compression methods (e.g., energy-aware pruning [37] and Deep compression [38]) directly cut down calculation tasks to slash overall energy consumption. This means the investigation of DNN calculation energy cost is still beneficial in energy-aware network light-weighting (e.g., using binary neural networks and doing network pruning) in energy-sensitive DNN scenarios, optimizing neuron structure (e.g., efficient activation functions) and designing energy-efficient application-specific integrated circuits (e.g., TPU) for DNN. The maximum energy efficiency of deep learning models can only be achieved through combined optimization of both computation and data movement energy consumption. We provide a review of DNN energy optimization, if interested, please refer to more details in supplementary material files.

C. Related Energy Quantification Research

There are several works investigating the energy consumption quantification and estimation of machine learning energy consumption. In [41], the authors make a survey on machine learning energy consumption estimation approaches that use simulated hardware or performance monitoring counters (PMC). They find processor plays the main role rather than DRAM in the energy consumption of tree-based models with experimentation. As an extended work, in [42], hardware energy observation tools and state-of-the-art machine learning energy consumption estimation approaches are reviewed and classified according to different techniques they use. Authors in [43] proposed a lightweight code-level energy estimation framework for software applications with limited additional cost in resources and energy. However, it is a general method focused on achieving accurate energy observation of computational hardware without the perspective of learning models.

Synergy, a method proposed in [39] uses linear regression on both the number of SIMD instructions and bus accesses observed from hardware PMC to measure and predict the energy consumption of CNNs at a layer-level. But no theoretical analysis of the relationship between CNN layer configuration and energy consumption is given. *NeuralPower* proposed in [40] uses sparse polynomial regression method to model the power and run-time according to layer configuration parameters (e.g., batch size, kernel size, etc.) of key CNN layers, then applies the model to unseen layers for energy consumption estimation. However, the layer-level analyze can not distinguish linear and non linear operations in layer configuration. Theoretical analysis of the relationship between CNN layer configuration and energy consumption is not given. Floating-point operations (FLOPs) is used as a simple model-structure-based DNN computational complexity measurement in [14] without mapping to DNN energy consumption. The calculation of FLOPs only consider linear floating-point operations (multiplication and accumulation) in full-connection/convolutional layers without non-linear operations and other layers (e.g., non-linear operations in AFs and operations in pooling layers). These ignored parts count a non-negligible part of DNN model complexity (around *5%* in convolutional layers and *15%* in feed-forward layers depends on the model configuration measured by our theoretical TOs model). Authors in [33] propose a theoretical DNN energy estimation method that uses the simulation of data movements between multiple layers of storage to quantify and normalize DNN energy consumption with the energy for doing one MAC operation as the unit. However, their method ignore non-linear operations and the difference between hardware (energy for doing one MAC operating varies in different computation hardware, the rate of energy consumption for storage and computation varies with different hardware combinations). No practical energy data is given, and the memory simulation method is not open to readers. As an extension work, they propose *Eyeriss* in [35], which optimizes the data flow of CNN models using higher data reuse rates and less data movement from expensive storage. Authors in [44] review the parameter size, FLOPs and performance metric of several benchmark DNN models. They find the computation and energy efficiency of hardware is affect by the precision of floating-point (FP) numbers (e.g., FP16, FP32), and propose that multiplicative factors take the majority responsibility of DNN energy consumption. However, the work is observation-based without study the energy scaling law led by DNN model configurations.

Summary	Precision (%)	Theory Basis	Experiment Data	Focus Area	Data Movement Energy	Calculation Energy	Energy Metric	Disadvantages
Experimentation of System Level Energy Consumption Increase [39]	63.05-73.7	×	✓	SIMD & No. Data Access	✓	No distinction of operation types	Joule	No analysis of the relationship between the DNN model configuration and energy consumption
Experimentation Layer-based DNN Energy Consumption and Core Usage [40]	97.21 (avg)	×	✓	Total Run-time Power	✓	No distinction of operation types	Joule	Not distinguishing linear and non-linear operations in layer configuration and energy consumption, need information about hardware run time and power
Theoretical Analysis of DNN Complexity based on Only Linear Operations [14]	92.9-99.5	✓	×	Calculation FLOPs	×	Linear Operations	FLOPs	Ignored Non-linear Operations (e.g., activation functions)
Theoretical & Experimental Analysis of DNN Energy Consumption based on Data Movements [33]	96.4-96.5	✓	✓	Data Movement & MACs	✓	Linear Operations	MACs	Data sparsity is hard to estimate accurately; ignored non-linear operations
This Paper: Proposed Transistor Operations (TOs) Method	93.6-99.5	✓	✓	Transistor Level Operations	×	Linear and non-linear operations	TOs/Joule	No data movement energy analysis

TABLE II
COMPARISON BETWEEN STATE-OF-THE-ART AND OUR PROPOSED METHODS FOR DNN ENERGY CONSUMPTION ESTIMATION. ACRONYMS IN TABLE: DEEP NEURAL NETWORK (DNN), SINGLE INSTRUCTION MULTIPLE DATA (SIMD), TRANSISTOR OPERATIONS (TOS), FLOATING-POINT OPERATIONS (FLOPS), MULTIPLY–ACCUMULATE OPERATIONS (MACS).

The comparison between aforementioned DNN energy estimation methods and our proposed TOs method are summarised in Table II, which also list the individual precision of each method.

D. Gap Summary & Innovation

Currently, as we have seen from the above review, the relationship between the DNN model configuration and energy consumption is not well established. The energy consumption of nonlinear operations in DNN is still lacking in analysis.

In this paper, we develop an innovative bottom-level Transistor Operations (TOs) method to expose the role of nonlinear activation functions and neural network structure in energy consumption. We translate a range of feedforward DNN and CNN models into ALU calculation tasks (e.g., basic operations (BOs)). Based on our TOs model, we demonstrate how the calculation energy scales when changing the model structure and activation functions. We also provide a verification experiment that compares the energy consumption estimated with TOs model and the practical energy consumption monitored from general-purpose commercial processors (CPU, GPU).

Compared with energy consumption estimation methods in [40], [42], our TOs method can individually analyze the calculation energy consumption with DNN model structures, and consider non-linear operations that not included in [14], [33], [40]. We will show that our proposed TOs method can achieve a superior 93.61% - 99.51% precision in estimating its energy consumption.

E. Research Limitations & Applicability

The calculation tasks measured by TOs/FLOPs/MAC have limitations in mapping DNN energy consumption when the DNN model runs with multi-core. The processing of data instances in DNN training follows the same way, suitable for parallel processing [45] in modern multi-core processors. However, plenty of additional energy costs for core communications will be generated [46]. This additional core communication cost is hard to split from overall processor energy consumption and does not influence DNN calculation tasks and complexity metrics (e.g., TOs, FLOPs). Simultaneously, as reported in [47], the core communication energy cost in multi-core processors is hard to model and is an experimental fact. Based on the aforementioned points, we only consider single thread & CPU running in this paper. In fact, this is a common issue for all theoretical metrics for DNN calculation tasks. At the same time, recent TOs model does not support piecewise-defined activation functions (e.g., Rectified Linear Unit (ReLU)) processed with comparison operators. As TOs is a theoretical metric that is directly analysed from DNN structures and configurations, it is not applicable to black box DNN models without structure and configuration information.

Ultimately, the practical energy of running the same DNN models on different hardware varies. As such, our proposed TOs model in this paper calculates the theoretical TOs for each hardware operation with generic processing logic. We will show it still outperforms FLOPs in estimating the energy consumption of DNN models on different hardware platforms.

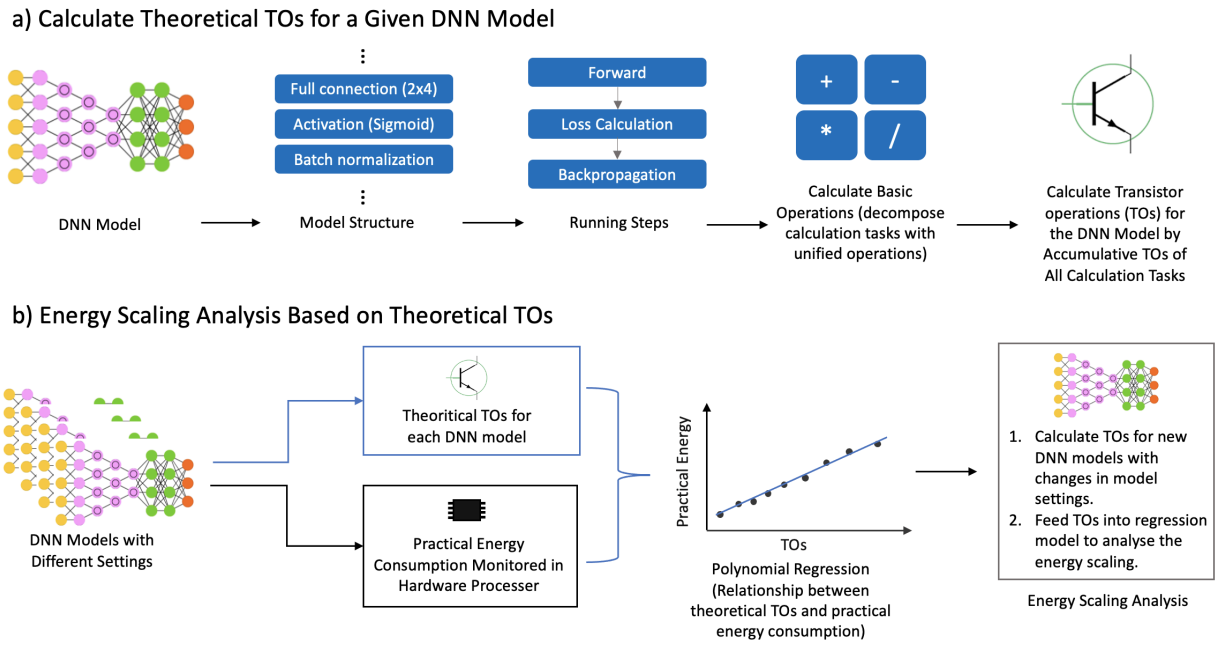


Fig. 3. Flowchart for Calculating Theoretical TOs of a Given DNN Model and DNN Energy Scaling Analysis. (a) Calculate TOs for a given model (left to right): extract layer structure from the target DNN model; define which steps are going to be analyzed; layer-wise analyze how much BOs are needed in forward/loss-calculation/backpropagation; translate layer-wise BOs into TOs. (b) Energy Breakdown of DNN Models (left to right): prepare a number of different DNN models; apply TOs model to calculate theoretical TOs of each DNN model, collect their practical energy consumption from hardware respectively; use polynomial regression (PR) to analyze the relationship between DNN model TOs and energy consumption for current hardware; input the TOs of new target DNN models into the trained TOs-based PR model to predict their energy consumption

II. Transistor Operations (TOs) Model

The calculation of theoretical TOs for a given DNN model is demonstrated as Fig. 3a (left to right). Suppose the dataset have already been pre-processed (energy for data pre-processing is not considered in this paper, but our TOs method could be extended to data processing). Firstly, the layer list will be extracted from the model structure and settings (e.g., 2×4 full-connection layer, activation layer with Sigmoid AF). Secondly, extract running steps according to different analysis levels: *training level* involves model forward, loss calculation and backpropagation (BP); *validation level* involves forward and loss calculation; *inference level* only focus on model forward. The reason is that each step has individual calculation logic and resulting in different calculation tasks and energy consumption. Calculation tasks at different analysis levels are calculated by summing the calculation tasks of the relevant steps (e.g., validation level calculation tasks is calculated by summing calculation tasks for model forward and loss calculation). Thirdly, layer-wise analysis of how many calculation tasks are needed for each running step based on their calculation logic, in terms of BOs. We use addition, subtraction, multiplication, division and root operations as the categories of BOs, for the reason that higher-level calculations (e.g., activation of neuron with Sigmoid as AF function - see Function 3) are assembled by these five BOs at the software level. Finally, we analyze how many transistor operations are theoretically involved in each basic operation processed by ALU calculation logic (translate BOs into TOs), and calculate the layer-based TOs based on the layer-based BOs information and data types (e.g., FP-16, FP-32).

Suppose we have a target DNN model to be analyzed, the layer list extracted from q is $L = \{l_1, l_2, ..., l_i, ..., l_o\}$ (l_o is the output layer). C represent the basic operation calculator, which takes a model layer as input and calculate how many basic operations are needed in this layer. T represent the TOs translator, which takes the number of five basic operations as input, and calculate how many TOs are needed to process all the input basic operations. The calculation of theoretical TOs for DNN model q could be shown as equation 1.

$$\text{Target Model TOs} = \sum_{i=1}^{o} T(C(l_i)) \tag{1}$$

The functions used in BOs and TOs calculations are introduced in the following subsections, frequently used notations are listed in Table III.

A. Layer Structure Based Basic Operations (BOs)

The BOs of a DNN model are calculated according to the processing logic of each layer, related to Fig. 3a: *Calculate basic operations*, details are shown in Alg. 1. For each layer $l_i \in L$, the BOs calculator $C_{\text{f}}(l_i)$ and $C_{\text{bp}}(l_i)$ analyze the function of the layer l_i by its structure; translate the layer function into calculation tasks (refers to Fig. 1a - ALU Calculation Tasks); and further translate calculation tasks into the number of five BOs needed for layer forward and backpropagation respectively. $C_{\text{loss}}(l_{\text{output}})$ takes only the output layer l_{output} structure as input to calculate BOs for loss calculation. The calculated BOs information by $C_{\text{f/bp/loss}}$ will be stored and return as a list (shown in equation 2). Here, n with sub-index

TABLE III
LIST OF NOTATIONS

L	Layer list of DNN model
l_i, l_o	The ith/output layer of DNN model
C_f, C_{bp}, C_{loss}	BOs calculator for layer forwarding, backpropagation and loss calculation
$n_{add}, n_{sub}, n_{mul}, n_{div}, n_{root}$	Number of addition, subtraction, multiplication division and root operation
B_f, B_{bp}	Layer-wise basic operations list of DNN model
$b_{fk}, b_{bpk}, b_{loss}$	Basic operations of ith/output layer
$BOs_{Potential}$ $BOs_{Activation}$ $BOs_{Convolution}$	Represent the number of five basic operations for potential/activation/convolution calculation in a layer
x	Neuron input
w	Weight
ξ	Neuron potential
y	Neuron output
$bias$	Bias
c_{in}, c_{out}	Number of input/output channel
m	Convolutional layer output window width
k	Convolutional kernel width
add/sub/mul/ div/root	One addition/subtraction/multiplication/ division/root operation
I	Number of input dimension
O	Number of output dimension
$T, T_{add}, T_{sub}, T_{mul}, T_{div}, T_{root}$	Transistor operations calculator (sub-index: calculator for different operations)
D_f, D_{bp}	Layer-wise transistor operations list
$d_{fk}, d_{bpk}, d_{loss}$	Layer-wise transistor operations
$d_{f_total}, d_{bp_total}$	Total transistor operations for model forwarding/ backpropagation
d_{total}	Total transistor operations for model forwarding, loss calculation and backpropagation
p	Data type

add, sub, mul, div and root means the number of each basic operation (fixed order: addition, subtraction, multiplication, division and root)).

$$C_{f/bp/loss}(l) = [n_{add}, n_{sub}, n_{mul}, n_{div}, n_{root}] \tag{2}$$

Algorithm 1 shows how to extract layer-wise BOs information from the target DNN model, the algorithm will return: layer-wise forward BOs in B_f; layer-wise BP BOs in B_{bp}; and BOs for loss calculation b_{loss}. We provide two examples that calculate the forward BOs for a full-connection layer in feedforward DNN, and a convolutional layer in CNN respectively to show how BOs is analyzed from layer structure. The calculation of BOs for loss and model backpropagation is similar to model forward but follows their individual calculation logic.

1) Feedforward DNN: The calculation logic for full connection layers is proposed in [48]. Suppose full-connection layer $l_{Full\text{-}connection}$ have I inputs; O outputs; use Sigmoid as AF, the calculation of output y_j on input x, weight w and bias $bias$ could be demonstrated in equation. Here, ξ represents neuron output before activated by AF, and S is Sigmoid function. The BOs for forward on one data instance could be calculated by equation 4:

$$\begin{aligned} &\text{Potential of } j\text{-th Neuron:} \xi_j = \text{bias}_j + \sum_{i=1}^{I}(w_{i,j} \times x_i) \\ &\text{Sigmoid AF of } j\text{-th Neuron:} y_j = S(\xi_j) = 1/1 + e^{-\xi_j} \end{aligned} \tag{3}$$

$$\begin{aligned} &C_f(l_{Full\text{-}connection}) = BOs_{Potential} + BOs_{Activation} \\ &BOs_{Potential} = O \times I(\text{mul} + \text{add}) \\ &BOs_{Activation} = O \times (\text{sub} + \text{add} + \text{div} + \text{root}) \\ &C_f(l_{Full\text{-}connection}) = [(I+1)O, O, IO, O, O] \end{aligned} \tag{4}$$

Here, the counting of BOs is analyzed from the logic of equations. For example, the calculation of ξ_j in equation 3 involves an accumulation of multiplication results, and an addition operation for adding bias. The multiplication operation happens I times; addition operation by accumulation repeat $I-1$ times; add bias need one additional addition operation; the total operations are I multiplications and $(I-1)+1$ additions. The calculation of ξ will repeat the process above O times, so the BOs for linear operations are $O \times I$ multiplications and additions. In equation 4, 'mul + add' means 1 multiplication operation and 1 addition operation. The items in the return list from $C_f(l_{Full\text{-}connection})$ means the number of each basic operation needed for forwarding this layer (the order is detailed before).

2) CNN: The working process of convolutional layers could be seen as Fig. 4. Suppose m, k, c_{in}, c_{out} are the output window width, convolutional kernel size, number of input channels and number of output channels in convolutional layer $l_{Convolutional}$. With GELU (approximate as GELU$(x) = 0.5 \times x \times (1 + \text{Tanh}(sqrt(2/\pi) \times (x + 0.044715 \times x^3)))$ [29] in PyTorch) as AF, the convolutional layer forward BOs on one input instance (e.g., one image) could be calculated as Eq. 5:

$$\begin{aligned} &C_f(l_{Convolutional}) = BOs_{Convolution} + BOs_{Activation} \\ &BOs_{Convolution} = m^2 \times c_{out} \times c_{in} \times k^2(\text{mul} + \text{add}) \\ &BOs_{Activation} = m^2 c_{out}(\text{sub} + \text{add} + \text{div} + \text{root} + 2 \times \text{mul}) \\ &n_{add} = m^2 c_{out}(1 + c_{in}k^2) \\ &n_{sub} = m^2 c_{out} \\ &n_{mul} = m^2 c_{out}(2 + c_{in}k^2) \\ &n_{div} = m^2 c_{out} \\ &n_{root} = m^2 c_{out} \\ &C_f(l_{Convolutional}) = [n_{add}, n_{sub}, n_{mul}, n_{div}, n_{root}] \end{aligned} \tag{5}$$

B. Basic Operations, Data Type and Theoretical TOs

The BOs information of the target DNN model will be further decomposed into transistor level to unify the energy metric of different BOs with TOs. As shown in Alg. 2, TOs calculator $T(b, p)$ takes both the BOs extracted by Alg. 1 and the data type p used in DNN model as inputs. The reason is that at different data precisions, different numbers of transistor operations are required to do the same operation. T opens the design logic of ALU, analyze the theoretical TOs for each operation modules (e.g., 32-bit adder, 16-bit multiplier) with their individual integrated circuit (IC) designs (the analysis method is introduced in the following subsections). With the information of layer BOs, T is used to translate BOs into theoretical TOs for DNN model in different steps (e.g., validation). Function T could be universally applied on the

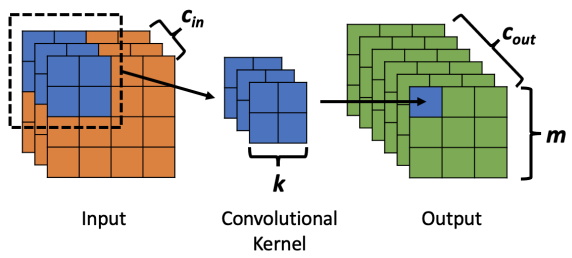


Fig. 4. Convolutional Layer

return lists of $C_{\text{f}}(l_i)$, $C_{\text{bp}}(l_i)$ and $C_{\text{bp}}(l_i)$ due to their similar data format (number of five basic operations with fixed order). Suppose b is a piece of BOs information extracted from the target DNN model by C, equation 6 demonstrate how b is processed into TOs with T. The calculation of DNN TOs from BOs information could be seen in Alg. 2, the algorithm will return layer-wise model forward TOs in D_{f} and BP TOs in D_{bp}, as well as cumulative TOs for model forward $d_{\text{f_total}}$, BP $d_{\text{bp_total}}$, loss d_{loss}, and overall TOs d_{total}.

$$\begin{aligned} b =& [n_{\text{add}}, n_{\text{sub}}, n_{\text{mul}}, n_{\text{div}}, n_{\text{root}}] \\ T(b,p) =& T_{\text{add}}(n_{\text{add}},p) + T_{\text{sub}}(n_{\text{sub}},p) + T_{\text{mul}}(n_{\text{mul}},p) \\ &+ T_{\text{div}}(n_{\text{div}},p) + T_{\text{root}}(n_{\text{root}},p) \end{aligned} \tag{6}$$

Algorithm 1 Calculate layer based BOs for DNN model

Require: DNN model structure $L = \{l_1, l_2, ..., l_{\text{o}}\}$
Require: Basic operation calculators: $C_{\text{f}}, C_{\text{bp}}, C_{\text{loss}}$
  Initialise $B_{\text{f}} = [b_{\text{f1}}, b_{\text{f2}}, ..., b_{\text{fo}}]$; $B_{\text{bp}} = [b_{\text{bp1}}, b_{\text{bp2}}, ..., b_{\text{bpo}}]$
  Initialise $b_{\text{loss}} = [0, 0, 0, 0, 0]$ /* number of 5 BOs */
  for $l_i \in [1, 2, ..., \text{o}]$ **do**
    $b_{\text{f}i} = C_{\text{f}}(l_i)$
    $b_{\text{bp}i} = C_{\text{bp}}(l_i)$
  end for
  $b_{\text{loss}} = C_{\text{loss}}(l_{\text{o}})$
  return $B_{\text{f}}, B_{\text{bp}}, b_{\text{loss}}$

1) BOs, Operation Units in ALU and Calculation of TOs: In TOs calculator, the calculation of TOs from BOs depends on the inner logic of ALU (open ALU and get number of logic gates, and open logic gates with transistors to count TOs). The various IC design for ALU inner logic follows the role of making a trade-off between time complexity (the time delay) and space complexity (number of transistors). The achieve of lower-delay ALU needs additional hardware logic gates which usually cost more transistors and energy [49] (e.g., adder: ripple-carry and carry look-ahead adder [50]; multiplier: Wallace and Booth-Wallace multiplier [51]). The logic of IC in ALU support different types of basic operations, however, not all the basic operations have their independent operation unit (e.g., adder is designed for addition, but subtraction is processed in adder by 2's complement). As multipliers are always built with adder as basic units, the changing of adder IC has a limited impact on the relationship between transistors in adder and multiplier, similar for other operation units. Although independent root operation units exist, they are not widely embedded in current PC processors. Root operation is always simulated with the Newton-Raphson method [52] by operation units in ALU. Our TOs model in this paper uses the basic IC design to calculate the theoretical transistors needed by BOs. Please note, the design of IC is not focused in this paper, using the specific hardware IC may increase the analysis accuracy of model energy scaling on that hardware. Theoretically, transistors needed in adder follows the linear relationship to bit-length, multiplier and divider follow exponential relationship. Ten transistors are used for a NOR-XNOR gates based 1-bit full-adder as proposed in [53]. And according to the IC design in [18], transistors used for the 64-bit Booth-Wallace multiplier and SRT divider are 90k and 110k respectively.

Algorithm 2 Calculate theoretical TOs based on BOs

Require: Layer based BOs set $B_{\text{f}} = [b_{\text{f1}}, b_{\text{f2}}, ..., b_{\text{fo}}]$; $B_{\text{bp}} = [b_{\text{bp1}}, b_{\text{bp2}}, ..., b_{\text{bpo}}]$
Require: BOs for loss calculation b_{loss}
Require: Data type p
Require: TOs calculator T
  Initialise $D_{\text{f}} = [d_{\text{f1}}, d_{\text{f2}}, ..., d_{\text{fo}}]$; $D_{\text{bp}} = [d_{\text{bp1}}, d_{\text{bp2}}, ..., d_{\text{bpo}}]$
  Initialise $d_{\text{f_total}}, d_{\text{bp_total}}, d_{\text{loss}}, d_{\text{total}} = 0, 0, 0, 0$
  for $i \in [0, 1, ..., \text{o}]$ **do**
    $d_{\text{f}i} = T(b_{\text{f}i}, p)$
    $d_{\text{bp}i} = T(b_{\text{bp}i}, p)$
    $d_{\text{f_total}} += T(b_{\text{f}i}, p)$
    $d_{\text{bp_total}} += T(b_{\text{bp}i}, p)$
  end for
  $d_{\text{loss}} = T(b_{\text{loss}}, p)$
  $d_{\text{total}} = d_{\text{f_total}} + d_{\text{bp_total}} + d_{\text{loss}}$
  return $D_{\text{f}}, D_{\text{bp}}, d_{\text{f_total}}, d_{\text{bp_total}}, d_{\text{loss}}, d_{\text{total}}$

2) Floating-point Numbers in TOs Calculation: Floating-point numbers (FP) are the most generally used data type in DNNs. As FP used in majority programming language follows IEEE 754 standard, the calculation of theoretical TOs should consider the structure and calculation logic of of binary FP with different precision (e.g., FP-32 known as single-precision floating-point, FP64 as double-precision floating-point). According to IEEE 754, each FP-32 number contains an 1-bit sign, an 8-bit exponent and a 23-bit fraction (FP-64: 1-bit sign, 11-bit exponent and 52-bit fraction). Theoretically, adding two FP-32 numbers will need a 24-bit adder (23 full-adder and 1 half-adder without considering bit shift in the exponent). As shown in Fig. 5, the multiplication/division of two FP-32 numbers will need a XOR gate (for sign), a 24-bit multiplier/divider (fraction calculation) and an 8-bit adder (exponent calculation). It means the TOs for a FP-32 multiplication is theoretically be calculated by the sum of transistors for a 24-bit multiplier, a 8-bit adder and a XOR gate. As a theoretical model, transistors redundancy in practical hardware are not considered (e.g., multiplication of two 24-bit number on 32-bit multiplier).

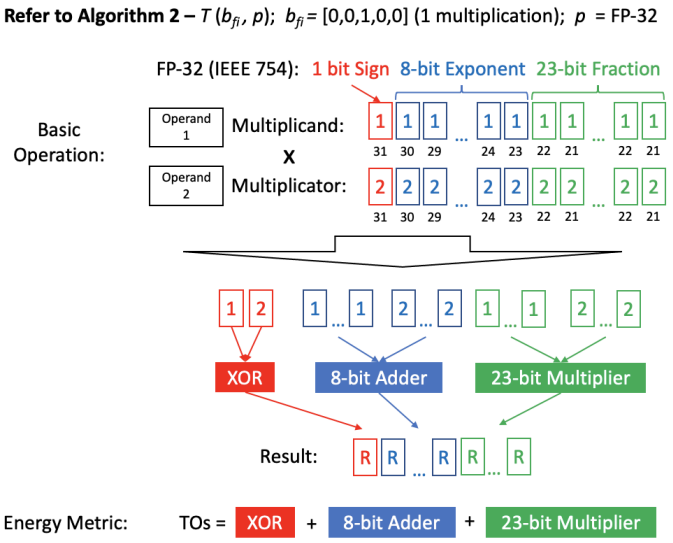


Fig. 5. Calculation of Theoretical TOs based on BOs (Example: Multiplication Operation on Two IEEE 754 FP-32 Numbers)

Exp. Settings	Feedforward DNN	CNN
Dataset	Banknote [54]	Drone images
Dataset Length	1372 instances	300 images
Data Structure	4 inputs & 1 output	256*256*3 (RGB)
Model Type	Feed-forward	Convolutional
Model Depth	3 hidden layers	2-10 C-layers
DNN Width	4-18 nodes per layer	-
CNN Kernel (h&w)	-	layer 2-6: 3
	-	layer 7-10: 4,5,6,7
CNN Channels	-	32
Activation Functions	Sigmoid, Tanh, GELU	GELU
Batch Size	64	64 (GPU: 16)
Epochs	2k (laptop CPU)	25 (laptop CPU)
	5k (desktop CPU)	20 (desktop CPU)
		100 (desktop GPU)
DNN Framework	PyTorch	PyTorch
Hardware	**Laptop**	**Desktop**
CPU	Intel Core i9	AMD RYZEN 9
GPU	-	Nvidia 3080
OS	macOS Monterey	Windows 10

TABLE IV
EXPERIMENT SETTINGS AND HARDWARE

C. TOs Model and Energy Scaling

Theoretical TOs for a given DNN model could be calculated by Alg. 1 and Alg. 2. The process of mapping the scaling of DNN model TOs to its practical energy consumption scaling is demonstrated in sub-figure Fig. 3*b*. Firstly, a list of DNN models with different configurations will be established. Secondly, calculate individual TOs of DNN models respectively with our TOs model, and deploy them on practical hardware for their practical energy consumption data. According to our theory, polynomial regression (PR) with different number of coefficients will be used to fit the relationship between the practical energy consumption and TOs of DNN models. To analyze the scaling of energy with a DNN design factor (e.g., width of hidden layers), a list of models will be generated by gradually changing the factor (e.g., models with 4,5,6 and 7 hidden layer width). Then, calculate their TOs and estimate their individual energy consumption by the previously fitted PR model. We demonstrate the energy scaling of a feedforward DNN model with different hidden layer widths and AFs in the next section, followed by that of a CNN model with different layer configurations. Please note that the practical energy consumption of models depends on different choices of hardware. If the hardware platform changes, the PR model should be retrained on energy data collected from the new hardware platform.

III. METHOD VERIFICATION

To verify the our TOs model, we design experiments to analyze the training energy scaling of 1) a feedforward DNN set by changing the AFs and width of hidden layers; 2) a CNN set by changing the number of convolutional layers and kernel size in each layer. The structural plausibility of DNN models (overfitting may occur with large network size) is not considered, as they do not affect energy consumption. The experiment settings could be seen from Table IV, and samples from used two datasets could be seen in Fig. 7 (data statement please refer to the supplementary material file).

The practical energy consumption of DNN models is collected with *Intel Power-Gadget software* [55] (for Intel Architecture CPU [56]), *OpenHardwareMonitor* (for AMD CPU) and *TechPowerUp GPU-Z* (for GPU). These tools are designed using on-chip energy sensors to collect the instantaneous power of processor cores, DRAM, and overall CPU/GPU package separately with timestamps [55]. The energy consumption is calculated based on the integration of the instantaneous power consumption over time. Certain errors could exist due to the accuracy of on-chip sensor for instantaneous readings reported in [57]. However, as our aim is to analyze only calculation energy cost, it is the only way for collecting on-chip component energy data [41], [58] and reported to be fairly accurate by NVIDIA (+-5% error rate [59]) and authors in [60], [61]. As the problem does not influence theoretical TOs/FLOPs, we will not discuss the accuracy of hardware energy monitoring. During the experiment, we force the program to run with a single CPU/GPU core as the reason mentioned in research limitations - core communications energy cost generated by parallel running could significantly influence the relationship between DNN calculation tasks and practical energy consumption.

Dataset: Banknote

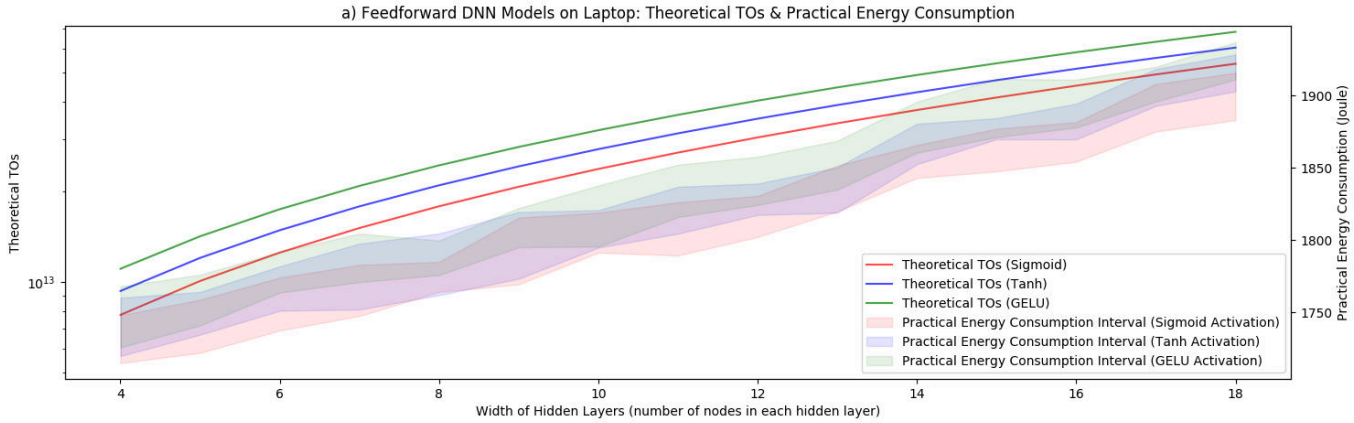
Feature 1	Feature 2	Feature 3	Feature 4	Label
3.6216	8.6661	-2.8073	-0.44699	0
4.5459	8.1674	-2.4586	-1.4621	0
3.866	-2.6383	1.9242	0.10645	0
3.4566	9.5228	-4.0112	-3.5944	0

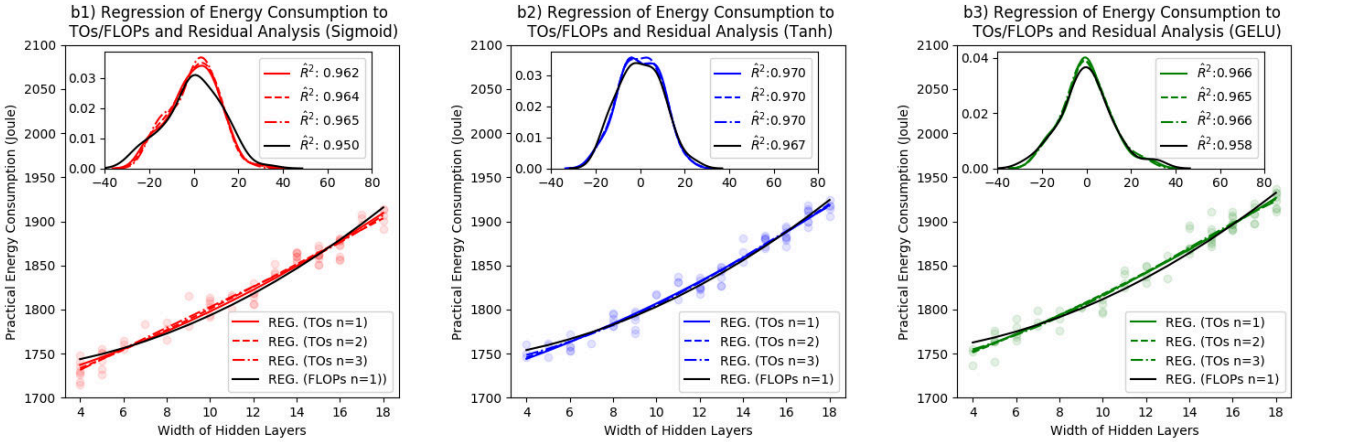
Dataset: Drone Images

Fig. 7. Used Dataset (Samples)

A. Energy Consumption Scaling of Feedforward DNNs on Laptop CPU

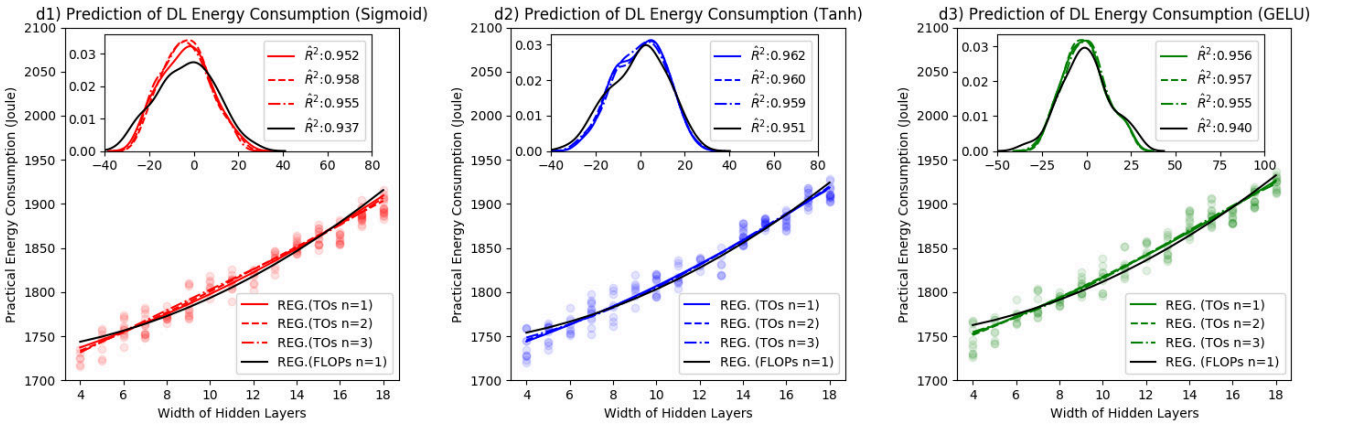
As shown in Fig. 6*a*, we calculate TOs for feed-forward DNNs with 4-18 width (number of nodes in each hidden layer) and use *Sigmoid*, *GELU* and *Tanh* as AFs respectively. We did 50 experiments (cool the hardware between each run





c) Training of Polynomial Regression (Energy to TOs/FLOPs): Coefficient and Performance Metrics

	n = 1	RMSE	$\hat{R}^2$	n = 2	RMSE	$\hat{R}^2$	n = 3	RMSE	$\hat{R}^2$
TOs (sigmoid)	1.71e+3, 3.65e-12	10.857	0.962	1.69e+3, 5.00e-12, -2.30e-26	10.480	0.964	1.69e+3, 5.55e-12, -4.31e-26, 2.15e-40	10.332	0.965
FLOPs (sigmoid)	1.73e+3, 8.42e-8	12.518	0.950	1.72e+3, 1.31e-7, -2.21e-17	10.852	0.962	1.71e+3, 1.64e-07, -5.69e-17, 1.03e-26	10.449	0.964
TOs (tanh)	1.71e+3, 3.34e-12	9.302	0.970	1.70e+3, 4.03e-12, -7.74e-27	9.273	0.970	1.71e+3, 3.23e-12, -1.80e-26, 2.43e-40	9.180	0.970
FLOPs (tanh)	1.74e+3, 8.62e-8	9.718	0.967	1.72e+3, 1.30e-7, -1.81e-17	9.178	0.970	1.72e+3, 1.39e-07, -2.76e-17, 2.83e-27	9.175	0.970
TOs (gelu)	1.72e+3, 3.04e-12	10.598	0.966	1.71e+3, 3.51e-12, -6.46e-27	10.567	0.965	1.73e+3, 2.01e-12, -3.59e-26, 3.53e-40	10.429	0.966
FLOPs (gelu)	1.75e+3, 8.84e-8	11.719	0.958	1.73e+3, 1.29e-7, -1.96e-17	10.454	0.966	1.73e+3, 1.31e-07, -2.19e-17, 6.88e-28	10.440	0.966



e) Validation of Polynomial Regression (Energy to TOs/FLOPs): Performance Metrics

	RMSE (n=1)	$\hat{R}^2$ (n=1)	RMSE (n=2)	$\hat{R}^2$ (n=2)	RMSE (n=3)	$\hat{R}^2$ (n=3)
TOs (sigmoid)	11.632	0.952	10.865	0.958	11.157	0.955
FLOPs (sigmoid)	13.349	0.937	10.921	0.957	11.090	0.956
TOs (tanh)	11.146	0.962	11.313	0.960	11.408	0.959
FLOPs (tanh)	12.610	0.951	11.403	0.960	11.427	0.959
TOs (gelu)	11.731	0.956	11.555	0.957	11.779	0.955
FLOPs (gelu)	13.610	0.940	11.731	0.955	11.853	0.954

Fig. 6. Method Verification (Feedforward DNNs on Laptop CPU)

to maintain the repeatability of experiment) on each model setting with laptop CPU as introduced in Table IV, drop the highest and lowest 5 data, and demonstrate the practical energy consumption interval of each DNN. We randomly split the collected energy data for each DNN into training sets (60%) and validation sets (40%). They are used to train and validate the PR models mapping the relationship between DNN TOs/FLOPs and practical energy consumption. We demonstrate the training of PR models based on FLOPs (FLOPs-based PR model) and TOs (TOs-based PR model) and with 1-3 coefficients respectively. We conduct a residual analysis to different PR models, the result could be seen in Fig. 6*b1-b3* (we only demonstrate 1 coefficient FLOPs-based PR in figures for higher clearness and readability, statistics for 2/3 coefficients FLOPs-based PR are listed in on-figure tables). In these figures, each point is one data instance collected from one experiment. We could see both FLOPS-based and TOs-based PR models perform fairly excellent - their residuals following Gaussian distribution. To provide statistical performance analysis of PR models, we calculate adjusted R-square and RMSE as shown in Fig. 6*c* (will be explained in Result Discussion). We further demonstrate the validation of PR models in Fig. 6*d1-d3*, and list the performance metric in Fig. 6*e*. We also run the DNN set on a desktop CPU, please refer to Fig. 1 in the supplementary material file.

According to the coefficients shown in Fig. 6*c* and statistic of DNN on desktop CPU (Fig. 1*c* in the supplementary material file), use data of Sigmoid with 1 coefficient (*n=1*) for example, the relationship between DNN model's TOs and the practical energy consumption *E* in current laptop CPU and desktop CPU could be demonstrated by Eq.7.

$$\begin{aligned} &\text{DNN on laptop CPU:}\\ &\mathrm{E} = 1.71 \times 10^{3} + 3.65 \times 10^{-12} \times \mathrm{TOs}\\ &\mathrm{E} = 1.73 \times 10^{3} + 8.42 \times 10^{-8} \times \mathrm{FLOPs}\\ &\text{DNN on desktop CPU:}\\ &\mathrm{E} = 9.96 \times 10^{2} + 2.47 \times 10^{-12} \times \mathrm{TOs}\\ &\mathrm{E} = 1.04 \times 10^{3} + 5.71 \times 10^{-8} \times \mathrm{FLOPs} \end{aligned} \tag{7}$$

B. Energy Consumption Scaling of CNN on Desktop GPU

As shown in Fig. 8*a*, we calculate TOs for CNN models with different depths (number of convolutional layers) and apply *GELU* as the AF for each convolutional layer. The configuration of each CNN model could be seen from Table IV. We did 50 experiments on each model, drop the highest and lowest 5 data, and demonstrate the practical energy consumption interval of each model. The energy data is split randomly into training set and validation set with a rate of $60/40\%$. We demonstrate the training and validation of PR energy models with residual analysis in Fig. 8*b-c*. We also summarise the PR model performance metric (adjusted R-square and RMSE) in Fig. 8*d-e*. We also run the same CNN set on a desktop CPU and a laptop CPU separately, please refer to Fig. 2 and Fig. 3 in the supplementary material file for your interest.

According to the coefficients showed in Fig. 8*c* and statistic of CNN on desktop CPU (Fig. 2*c* in the supplementary material file), the relationship (PR models with *n=1*) between CNN model's TOs and the practical energy consumption *E* in current desktop processors could be demonstrated by Eq.8.

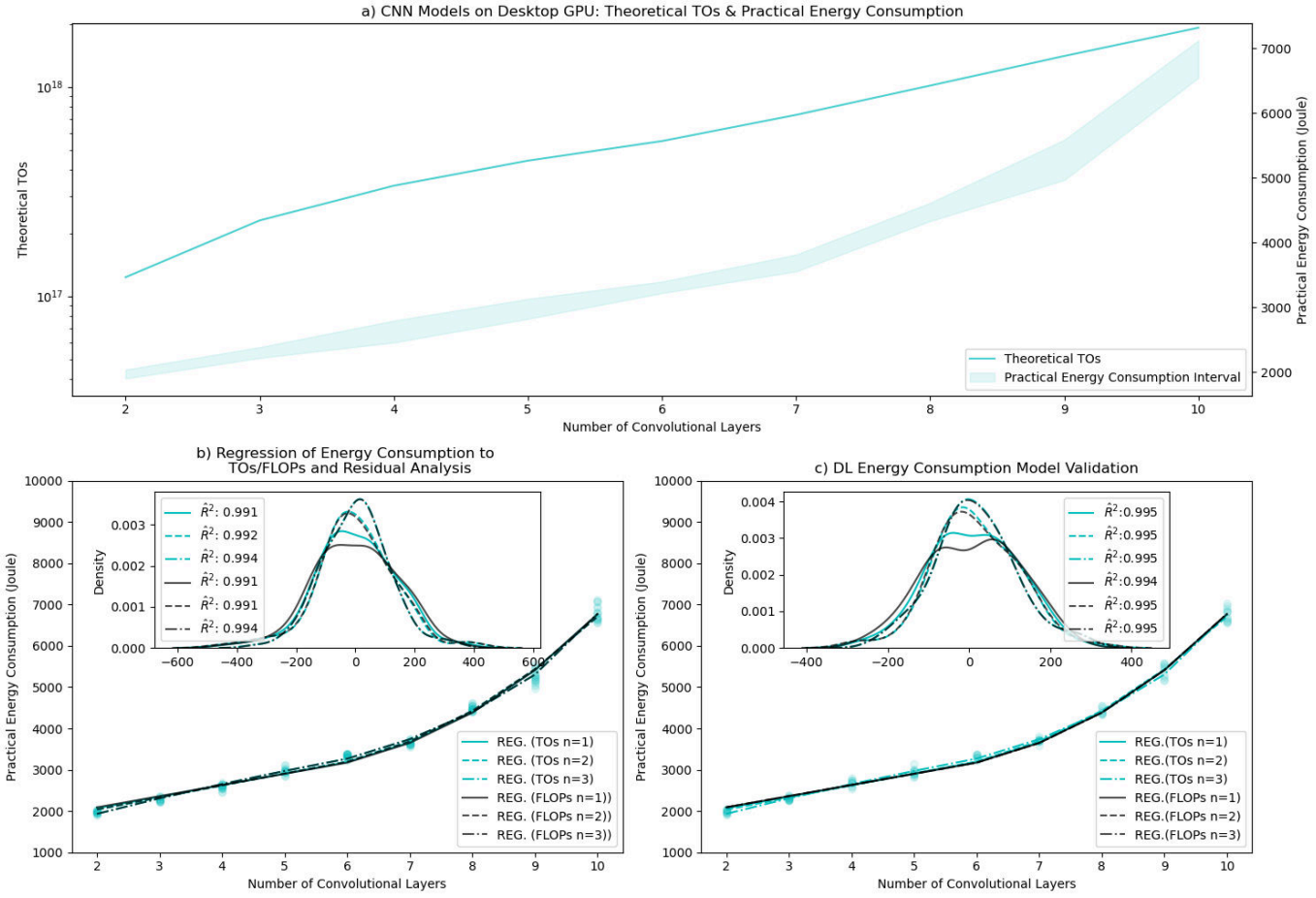
$$\begin{aligned} &\text{CNN on desktop GPU:}\\ &\mathrm{E} = 1.76 \times 10^{3} + 2.61 \times 10^{-15} \times \mathrm{TOs}\\ &\mathrm{E} = 1.81 \times 10^{3} + 3.49 \times 10^{-11} \times \mathrm{FLOPs}\\ &\text{CNN on desktop CPU:}\\ &\mathrm{E} = 2.51 \times 10^{3} + 4.29 \times 10^{-14} \times \mathrm{TOs}\\ &\mathrm{E} = 2.67 \times 10^{3} + 5.73 \times 10^{-10} \times \mathrm{FLOPs} \end{aligned} \tag{8}$$

C. Result Discussion

During the verification, each experiment with individual settings (feedforward DNNs/CNNs on different hardware processors) is run 50 times, to find the practical energy consumption interval and analyze the robustness of energy scaling models (see - sub-figure a in Fig. 6, 8 and Fig. 1, 2, 3 in the supplementary material file).

We analyze and compare the performance of TOs-based and FLOPs-based PR models in estimating DNN energy consumption by Adjusted R-square Error ($\hat{R}^2$) and Root Mean Square Error (RMSE). $\hat{R}^2$ represents the fraction of variance of the actual value of the response variable captured by the regression model, with penalty in the number of variables [62]. RMSE represents the differences between values predicted by a model or an estimator and the values observed [63]. From the statistic of $\hat{R}^2$ and RMSE (see sub-figures c, e in Fig. 6, and d, e in Fig. 8), both TOs-based and FLOPs-based PR models could accurately fit DNN energy metric and practical energy consumption ($\hat{R}^2$ for FLOPs/TOs-based PR model: 0.95-0.99/0.96-0.99), while TOs-based PR models have equal or better performance in $\hat{R}^2$ and RMSE than FLOPs-based PR models. This means, TOs is more accurate to be used as a metric for analysing the scaling law of DNN energy consumption. As we apply polynomial regression with 1-3 coefficients to fit the relationship between FLOPs/TOs and practical energy consumption, according to $\hat{R}^2$, sometimes the relationship shows more linear features and is more explainable. For example, as in function 7, 8, we can approximate the energy cost of each TO in CNN models on the GPU is an order of magnitude lower than that of CPU (desktop GPU: 2.61×10^{-15} Joule; desktop CPU: 4.29×10^{-14} Joule). Simultaneously, the performance of FLOPs-based and TOs-based PR models for DNN energy estimation in Joule could be summarised in Table V. Compared with FLOPs-based PR models, TOs-based PR models achieve $0.14 - 2.56\%$ higher precision on DNN energy consumption estimation, and a 10% lower average estimation error in Joule.

We also demonstrate DNN energy consumption over performance in Mean Squared Error (MSE) with different configurations (banknotes identification [54]) in Fig. 9, from where the energy efficiency of DNN model configurations (number of nodes and the choice of AFs) could be analyzed. From



d) Training of Polynomial Regression (Energy to TOs/FLOPs): Coefficient and Performance Metrics

	n = 1	RMSE	$\hat{R}^2$	n = 2	RMSE	$\hat{R}^2$	n = 3	RMSE	$\hat{R}^2$
TOs	1.76e+3, 2.61e-15	134.266	0.991	1.69e+3, 2.82e-15, -1.04e-34	131.907	0.992	1.53e+3, 3.74e-15, -1.29e-33, 4.00e-52	111.769	0.994
FLOPs	1.81e+3, 3.49e-11	139.645	0.991	1.72e+3, 3.89e-11, -2.68e-26	134.598	0.991	1.55e+3, 5.19e-11, -2.56e-25, 1.05e-39	111.672	0.994

e) Validation of Polynomial Regression (Energy to TOs/FLOPs): Performance Metrics

	RMSE (n=1)	$\hat{R}^2$ (n=1)	RMSE (n=2)	$\hat{R}^2$ (n=2)	RMSE (n=3)	$\hat{R}^2$ (n=3)
TOs	107.923	0.995	103.700	0.995	100.391	0.995
FLOPs	113.994	0.994	105.624	0.995	100.812	0.995

Fig. 8. Method Verification (CNN on Desktop GPU)

the figure, DNNs with Tanh as AF can converge to lower MSE loss within a certain energy cost than with GELU and Sigmoid (e.g., Tanh/GELU/Sigmoid can achieve: 0.004, 0.008, 0.013 MSE loss in 0.2×10^{14} TOs). At the same time, with AF fixed, the energy cost to train a DNN model to a certain MSE loss increases with network size (e.g., DNN with 4/8/12 width need: $0.17/0.32/0.45 \times 10^{14}$ TOs to 0.005 MSE loss). However, larger DNNs can converge to a lower MSE loss than light networks, with a significantly increased energy cost.

FLOPs/MACs/TOs are theoretical metrics for DNN complexity/energy consumption analysed from DNN structures and configurations. With support to nonlinear operations, TOs is more complex to be calculated than FLOPs/MACs. Compared with methods in [39], [40], TOs method is more efficient due to no practical experiment required, but not applicable to black box models.

IV. Conclusion and Further Works

The energy consumption of nonlinear operations in DNNs has not been well analyzed and modelled, resulting in an incomplete understanding of how DNN energy consumption scales with model complexity. In this paper, we propose

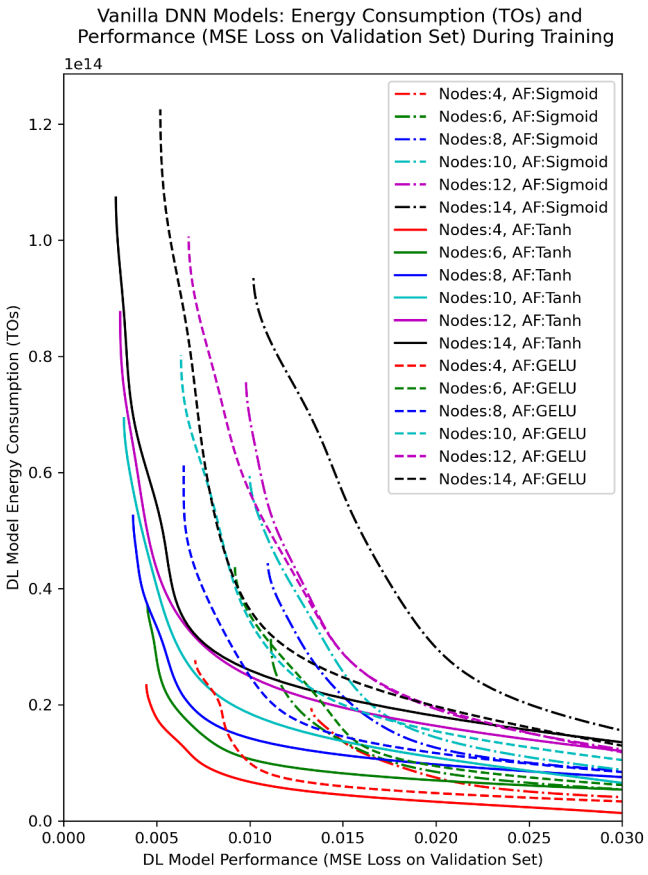


Fig. 9. TOs Energy Consumption and Performance of Feedforward DNNs

Desktop		
DNN: AFs	**FLOPs**	**TOs**
Sigmoid (%)	96.93-99.99 (avg 99.05)	97.46-99.99 (avg 99.14)
Tanh (%)	96.81-99.97 (avg 99.04)	96.95-99.99 (avg 99.22)
GELU (%)	97.11-99.97 (avg 98.94)	97.52-99.99 (avg 99.26)
Sigmoid (avg/max, J)	11.16/36.05	10.19/31.29
Tanh (avg/max, J)	11.33/41.28	9.33/39.51
GELU (avg/max, J)	12.78/35.55	9.04/29.45
CNN: Processor		
CPU (%)	78.41-99.92 (avg 94.13)	80.97-99.97 (avg 94.82)
GPU (%)	91.50-99.92 (avg 97.15)	92.61-99.98 (avg 97.36)
CPU (avg/max, J)	383.39/797.24	341.46/747.77
GPU (avg/max, J)	101.65/364.33	95.95/362.66
Laptop		
DNN: AFs	**FLOPs**	**TOs**
Sigmoid (%)	98.14-99.99 (avg 99.41)	98.52-99.99 (avg 99.49)
Tanh (%)	97.95-99.98 (avg 99.45)	98.44-99.99 (avg 99.50)
GELU (%)	97.99-99.99 (avg 99.46)	98.40-99.99 (avg 99.51)
Sigmoid (avg/max, J)	10.59/31.80	9.31/25.56
Tanh (avg/max, J)	9.90/35.27	9.19/26.77
GELU (avg/max, J)	9.85/34.57	9.19/30.47
CNN: Processor		
CPU (%)	74.03-99.90 (avg 92.93)	76.45-99.99 (avg 93.61)
CPU (avg/max, J)	501.09/1134.58	453.25/1029.06

TABLE V
PERFORMANCE OF FLOPS-BASED AND TOS-BASED PR MODELS (PRECISION, AVERAGE ERROR AND MAX ERROR)

a bottom-up theoretical TOs method to expose the role of nonlinear activation functions and neural network structure in DNN energy consumption. We show that 1) with single core running, theoretical TOs of DNN shows a strong empirical polynomial relationship with its practical energy, and could be used for analysing the energy scaling of DNN models; 2) the proposed method (average precision 93.61-99.51%) outperforms FLOPs-based method with $0.14 - 2.56\%$ higher precision on DNN energy consumption estimation, and lower 10% of the average estimation error; and 3) our scaling relationships are less prone to measurement errors than absolute energy consumption estimates.

The impact of our proposed TOs-based approach is that developers can analyze the energy scaling of different operations in DNNs, thus developing more energy-efficient DNN structures and configurations. We believe TOs could be extended to all DNNs through more comprehensive research in different algorithm logic (e.g., automatic differentiation in PyTorch [64]) and processing mechanism of bottom-level operations, e.g., comparison operators.

In future work, if we combine our proposed TOs-based and the data movement-based [33] energy estimation methods, we can build a more holistic and accurate DNN energy consumption framework. Furthermore, by combining our proposed TOs with state-of-the-art multi-core energy consumption modelling approaches, we can map the scaling law of DNN energy consumption in multi-core environments. We intent to apply this to machine learning techniques used in widespread communication [7] and IoT architectures [3] to have a significant impact.

REFERENCES

[1] D. Amodei, D. Hernandez, G. Sastry, J. Clark, G. Brockman, and I. Sutskever, "Ai and compute," *Heruntergeladen von https://blog. openai. com/aiand-compute*, 2018.

[2] N. Jones, "The information factories," *Nature*, vol. 561, 2018.

[3] W. Guo, S. Zhou, Y. Chen, S. Wang, X. Chu, and Z. Niu, "Simultaneous information and energy flow for iot relay systems with crowd harvesting," *IEEE Communications Magazine*, vol. 54, 2016.

[4] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Systems Journal*, vol. 10, 2016.

[5] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in nlp," *Annual Meeting of the ACL*, 2019.

[6] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Compute and energy consumption trends in deep learning inference," *arXiv preprint arXiv:2109.05472*, 2021.

[7] Z. Du, Y. Deng, W. Guo, A. Nallanathan, and Q. Wu, "Green deep reinforcement learning for radio resource management: Architecture, algorithm compression, and challenges," *IEEE Vehicular Technology Magazine*, vol. 16, no. 1, pp. 29–39, 2021.

[8] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao, "You only learn one representation: Unified network for multiple tasks," *arXiv preprint arXiv:2105.04206*, 2021.

[9] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[10] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green ai," *Communications of the ACM*, vol. 63, no. 12, pp. 54–63, 2020.

[11] E. Strickland, "Andrew Ng, AI Minimalist: The Machine-Learning Pioneer Says Small is the New Big," *IEEE Spectrum*, vol. 59, 2022.

[12] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, 2020.

[13] B. Li, P. Chen, H. Liu, W. Guo, X. Cao, J. Du, C. Zhao, and J. Zhang, "Random sketch learning for deep neural networks in edge computing," *Nature Computational Science*, vol. 1, no. 3, pp. 221–228, 2021.

[14] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv preprint arXiv:1611.06440*, 2016.

[15] V. Camus, C. Enz, and M. Verhelst, "Survey of precision-scalable multiply-accumulate units for neural-network processing," in *IEEE Int. Conf. on Artificial Intelligence Circuits and Systems*, 2019.

[16] J. Yu, J. Park, S. Park, M. Kim, S. Lee, D. H. Lee, and J. Choi, "Nn-lut: neural approximation of non-linear operations for efficient transformer inference," in *ACM/IEEE Design Automation Conference*, 2022.

[17] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, 2014.

[18] S. Kuninobu, T. Nishiyama, H. Edamatsu, T. Taniguchi, and N. Takagi, "Design of high speed mos multiplier and divider using redundant binary representation," in *IEEE Symposium on Computer Arithmetic*, 1987.

[19] H. Qin, R. Gong, X. Liu, X. Bai, J. Song, and N. Sebe, "Binary neural networks: A survey," *Pattern Recognition*, vol. 105, p. 107281, 2020.

[20] Q. H. Vo, N. L. Le, F. Asim, L.-W. Kim, and C. S. Hong, "A deep learning accelerator based on a streaming architecture for binary neural networks," *IEEE Access*, vol. 10, pp. 21 141–21 159, 2022.

[21] M. Christ, F. de Dinechin, and F. Pétrot, "Low-precision logarithmic arithmetic for neural network accelerators," in *IEEE Int. Conf. on Application-specific Systems, Architectures and Processors*, 2022.

[22] A. Sabbagh Molahosseini, L. Sousa, A. A. Emrani Zarandi, and H. Vandierendonck, "Low-precision floating-point formats: From general-purpose to application-specific," *Approximate Computing*, 2022.

[23] N. Wang, J. Nie, J. Li, K. Wang, and S. Ling, "A compression strategy to accelerate lstm meta-learning on fpga," *ICT Express*, 2022.

[24] S. M. Mishra, A. Tiwari, H. S. Shekhawat, P. Guha, G. Trivedi, P. Jan, and Z. Nemec, "Comparison of floating-point representations for the efficient implementation of machine learning algorithms," in *IEEE International Conference Radioelektronika*, 2022.

[25] A. Fawzi, M. Balog, A. Huang, T. Hubert, B. Romera-Paredes, M. Barekatain, A. Novikov, F. J. R Ruiz, J. Schrittwieser, G. Swirszcz *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[26] P. Pennestrì, Y. Huang, and N. Alachiotis, "A novel approximation scheme for floating-point square root and inverse square root for fpgas," in *International Conference on Modern Circuits and Systems Technologies*, 2022.

[27] N. Campos, E. Edirisinghe, S. Fatima, S. Chesnokov, and A. Lluis, "Fpga implementation of a custom floating-point library," in *SAI Intelligent Systems Conference*, 2023.

[28] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *IEEE International Solid-State Circuits Conference*, 2019.

[29] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[30] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman, *Compilers: principles, techniques, & tools*. Pearson Education India, 2007.

[31] D. G. A. Godse, *Digital Logic Design*. Technical Publications, 2009.

[32] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

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

[34] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE conference on computer vision and pattern recognition*, 2017.

[35] 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, 2016.

[36] A. Shah, C.-Y. Wu, J. Mohan, V. Chidambaram, and P. Krähenbühl, "Memory optimization for deep networks," *arXiv preprint arXiv:2010.14501*, 2020.

[37] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE conference on computer vision and pattern recognition*, 2017.

[38] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[39] C. F. Rodrigues, G. Riley, and M. Luján, "Synergy: An energy measurement and prediction framework for convolutional neural networks on jetson tx1," in *Int. Conference on Parallel and Distributed Processing Techniques and Applications*, 2018.

[40] 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*. PMLR, 2017, pp. 622–637.

[41] E. García-Martín, N. Lavesson, H. Grahn, E. Casalicchio, and V. Boeva, "How to measure energy consumption in machine learning algorithms," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2018, pp. 243–255.

[42] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[43] R. W. Ahmad, A. Naveed, J. J. Rodrigues, A. Gani, S. A. Madani, J. Shuja, T. Maqsood, and S. Saeed, "Enhancement and assessment of a code-analysis-based energy estimation framework," *IEEE Systems Journal*, vol. 13, no. 1, pp. 1052–1059, 2018.

[44] R. Desislavov, F. Martínez-Plumed, and J. Hernández-Orallo, "Compute and energy consumption trends in deep learning inference," *arXiv preprint arXiv:2109.05472*, 2021.

[45] J. J. Dai, Y. Wang, X. Qiu, D. Ding, Y. Zhang, Y. Wang, X. Jia, C. L. Zhang, Y. Wan, Z. Li *et al.*, "Bigdl: A distributed deep learning framework for big data," in *ACM Symposium on Cloud Computing*, 2019.

[46] M. J. Quinn, "Parallel programming," *TMH CSE*, vol. 526, p. 105, 2003.

[47] A. Lastovetsky and R. R. Manumachu, "New model-based methods and algorithms for performance and energy optimization of data parallel applications on homogeneous multicore clusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 4, pp. 1119–1133, 2016.

[48] D. Svozil, V. Kvasnicka, and J. Pospichal, "Introduction to multi-layer feed-forward neural networks," *Chemometrics and intelligent laboratory systems*, vol. 39, no. 1, pp. 43–62, 1997.

[49] Z. Abid, H. El-Razouk, and D. A. El-Dib, "Low power multipliers based on new hybrid full adders," *Microelectronics Journal*, vol. 39, 2008.

[50] J. Saini, S. Agarwal, and A. Kansal, "Performance, analysis and comparison of digital adders," in *IEEE Int. Conference on Advances in Computer Engineering and Applications*, 2015.

[51] S. Asif and Y. Kong, "Performance analysis of wallace and radix-4 booth-wallace multipliers," in *IEEE Electronic System Level Synthesis Conference*, 2015.

[52] M. A. Cornea-Hasegan, R. A. Golliver, and P. Markstein, "Correctness proofs outline for newton-raphson based floating-point divide and square root algorithms," in *IEEE Symposium on Computer Arithmetic*, 1999.

[53] H. T. Bui, Y. Wang, and Y. Jiang, "Design and analysis of low-power 10-transistor full adders using novel xor-xnor gates," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 1, pp. 25–30, 2002.

[54] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[55] S.-W. Kim, J. J.-S. Lee, V. Dugar, and J. De Vega, "Intel® power gadget," *Intel Corporation*, vol. 7, 2014.

[56] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem, "Introduction to intel core duo processor architecture." *Intel Technology Journal*, 2006.

[57] M. Fahad, A. Shahid, R. R. Manumachu, and A. Lastovetsky, "A comparative study of methods for measurement of energy of computing," *Energies*, vol. 12, no. 11, p. 2204, 2019.

[58] E. Calore, A. Gabbana, S. F. Schifano, and R. Tripiccione, "Thunderx2 performance and energy-efficiency for hpc workloads," *Computation*, vol. 8, no. 1, p. 20, 2020.

[59] N. Coporation, "Nvml api pages-for gpu utilization," 2017.

[60] S. Desrochers, C. Paradis, and V. M. Weaver, "A validation of dram rapl power measurements," in *Proceedings of the Second International Symposium on Memory Systems*, 2016, pp. 455–470.

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

[62] J. Miles, "R-squared, adjusted r-squared," *Encyclopedia of statistics in behavioral science*, 2005.

[63] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature," *Geoscientific model development*, vol. 7, 2014.

[64] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

https://dspace.org/

A transistor operations model for deep learning energy consumption scaling law

Li, Chen; Tsourdos, Antonios; Guo, Weisi

2024-01-01
Institute of Electrical and Electronics Engineers (IEEE)
https://doi.org/10.1109/TAI.2022.3229280
Li C, Tsourdos A, Guo W. (2024) A transistor operations model for deep learning energy consumption scaling law. IEEE Transactions on Artificial Intelligence, Volume 5, Issue 1, January 2024, pp. 192-204

Downloaded from a DSpace repository. https://dspace.org/