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 (DNNs) have transformed the automation of a wide range of industries and found 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 unrelated energy consumption anomalies, whilst the latter does not accurately reflect nonlinear computations. In this article, we are the first to develop a bottom-up transistor operations (TOs) approach to expose the role of nonlinear 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 on a variety of datasets and hardware configurations—achieving a 93.6%–99.5% precision. This outperforms existing floating-point operations (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 (300 000× 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 method to expose the important role of both (1) nonlinear activation functions and (2) DNN model structure in energy consumption. With our proposed TOs model, developers are able to analyze the energy efficiency of nonlinear activation functions, and make a trade-off between model performance and sustainability. The method can help developers predict energy consumption ahead of the practical run of DNN models on hardware to 93.61%–99.51% precision, which could reduce DNN experiments to save energy. Due to the consideration of both linear and non-linear operation in TOs, it can replace FLOPs/MACs to some extent as a more accurate metric of DNN model complexity.

Index Terms—Deep learning, energy consumption, model architecture, transistor operations (TOs).

Manuscript received 17 October 2022; revised 9 November 2022; accepted 10 December 2022. Date of publication 14 December 2022; date of current version 8 January 2024. This work was supported by the EC H2020 DAWN4IoE—Data Aware Wireless Network for Internet-of-Everything under Grant 778305. This article was recommended for publication by Associate Editor Mehmet Onder Efe upon evaluation of the reviewers’ comments. (Corresponding author: Weisi Guo.)

Chen Li and Antonios Tsourdos are with the Digital Aviation Research Technology Centre (DARTeC), Cranfield University, Bedford MK430AL, U.K. (e-mail: c.li.21@cranfield.ac.uk; a.tsourdos@cranfield.ac.uk).

Weisi Guo is with the Digital Aviation Research Technology Centre (DARTeC), Cranfield University, Bedford MK430AL, U.K., and also with the Alan Turing Institute, London NW12DB, U.K. (e-mail: weisi.guo@cranfield.ac.uk).

This article has supplementary downloadable material available at https://doi.org/10.1109/TAI.2022.3229280, provided by the authors.

Digital Object Identifier 10.1109/TAI.2022.3229280

I. INTRODUCTION

RAPIDLY increased artificial intelligence (AI) demand has generated a huge increase in computational resource requirement (300 000× 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-Everything will connect intelligence and it is important to reduce energy consumption across connectivity and autonomy [4]. The endless chasing of...
higher-precision in deep neural network (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 and robustness) and backgrounds (e.g., computational hardware and energy supply) of DNN tasks to make a tradeoff 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.

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 floating-point operations (FLOPs) [14] and MACs [15] only consider linear operations (e.g., multiplications and additions) without nonlinear operations (e.g., root operation in activation functions). However, nonlinear operations are a non-negligible part of some DNN models [16] that the amount of calculation tasks and energy cost for doing a nonlinear operation is always higher than linear ones [17], [18]. Thus, using FLOPs/MACs in analyzing 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], [23], [24], and high-efficiency operators [25], [26], [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, nonlinear operations, and floating-point (FP) 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 DNN 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 hyperparameters, 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]. First, operands will be called...
from data storage (dynamic random-access memory (DRAM), etc.) to the processor registers through data bus. The energy for moving one unit data varies with the memory hierarchy. Second, do the specific hardware operation (specified in operator) on operands, and generate calculation energy consumption. Third, 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.

1) Analyze the origin of hardware energy consumption at transistor operation (TO) and data movement level.
2) Translate linear and nonlinear DNN activation functions into the real calculation tasks executed by ALU.
3) Propose a TO-based energy metric that is generalized across diverse hardware operations.
4) 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 the following 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 consumption, especially for large datasets [32], [33], [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 depend 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 [34] and deep compression [32]) 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 (ICs) (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 [39], 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 [40], 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 [41] 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 [37] uses linear regression on both the number of single instruction multiple data 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 [38] uses sparse polynomial regression (PR) 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 cannot distinguish linear and nonlinear operations in layer configuration. Theoretical & Experimental Analysis of DNN Energy Consumption based on Data Movements [33] uses a simple model-structure-based DNN computational complexity measurement in [14] without mapping to DNN energy consumption. The calculation of FLOPs only consider linear FLOPs (multiplication and accumulation) in full-connection/convolutional layers without nonlinear operations and other layers (e.g., nonlinear operations in activation functions (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 nonlinear 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 [6] 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 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.

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

### Table II

| 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 of 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 |

Acronyms in table: Deep neural network (DNN), Single instruction multiple data (SIMD), Transistor operations (TOs), Floating-point operations (FLOPs), Multiply–accumulate operations (MACs).

D. Gap Summary and Innovation

Currently, as we have seen from the abovementioned review, the relationship between the DNN model configuration and energy consumption is not well established. The energy
In this article, we develop an innovative bottom-level 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] and [38], our TOs method can individually analyze the calculation energy consumption with DNN model structures, and consider nonlinear operations that not included in [14], [33], and [38]. 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 and Applicability

The calculation tasks measured by TOs/FLOPs/MAC have limitations in mapping DNN energy consumption when the DNN model runs with multicore. The processing of data instances in DNN training follows the same way, suitable for parallel processing [42] in modern multicore processors. However, plenty of additional energy costs for core communications will be generated [43]. 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 [44], the core communication energy cost in multicore processors is hard to model and is an experimental fact. Based on the aforementioned points, we only consider single thread and CPU running in this article. 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) processed with comparison operators. As TOs are a theoretical metric that is directly analyzed 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 article 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.
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). Third, 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 multiplication, division, and root operations as the categories of BOs, for the reason that higher level calculations (e.g., activation functions for DNN model are calculated according to the calculation logic for full connection BOs, and a convolutional layer in CNN, respectively, to show how BOs needed for layer forward and BP, respectively. BOs for loss calculation. The calculated BOs information by \( C_{\text{loss}}(1) \) will be stored and return as a list (as shown in (2)). Here, \( n \) with subindex add, sub, mul, div, and root means the number of each BO (fixed order: addition, subtraction, multiplication, division, and root).

\[
C_{\text{loss}}(1) = [n_{\text{add}}, n_{\text{sub}}, n_{\text{mul}}, n_{\text{div}}, n_{\text{root}}].
\]

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 \( b_{\text{loss}} \). 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 BP is similar to model forward but follows their individual calculation logic.

1) Feedforward DNN: The calculation logic for full connection layers is proposed in [45]. Suppose full-connection layer \( l_{\text{full-connection}} \) have \( I \) inputs; \( O \) outputs; use Sigmoid as AF, the calculation of output \( y_j \) on input \( x \), weight \( w \), and bias \( b \) could be demonstrated in equation. Here, \( \xi \) represents neuron output before activated by AF, and \( S \) is Sigmoid function. The BOs for forward on one data instance could be calculated by (4).

\[
\text{Potential of } j-\text{th Neuron: } \xi_j = \text{bias}_j + \sum_{i=1}^{O} (w_{i,j} \times x_i).
\]

Sigmoid AF of \( j \)-th Neuron: \( y_j = S(\xi_j) = 1/1 + e^{-\xi_j} \)

\[
C_f(l_{\text{full-connection}}) = \text{BOsPotential} + \text{BOsActivation}
\]

\[
\text{BOsPotential} = O \times (\text{mul} + \text{add})
\]

\[
\text{BOsActivation} = O \times (\text{sub} + \text{add} + \text{div} + \text{root})
\]

\[
C_f(l_{\text{full-connection}}) = [(I + 1)O, O, IO, O, O].
\]

Here, the counting of BOs is analyzed from the logic of equations. For example, the calculation of \( \xi_j \) in (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 operation in total; the operations are $I$ multiplications and $(I-1)+1$ additions. The calculation of $\xi$ will repeat the process above $O$ times, so the BOs for linear operations are $O \times I$ multiplications and additions. In (4), “mul + add” means 1 multiplication operation and 1 addition operation. The items in the return list from $C_l(l_{\text{Convolutional}})$ means the number of each BO 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 $m$, $k$, $c_{\text{in}}$, $c_{\text{out}}$ are the output window width, convolutional kernel size, number of input channels, and number of output channels in convolutional layer $l_{\text{Convolutional}}$. With GELU (approximate as GELU in log (the order is detailed before).

$C_l(l_{\text{Convolutional}}) = \text{BOs}_{\text{Convolution}} + \text{BOs}_{\text{Activation}}$

$\text{BOs}_{\text{Convolution}} = m^2 \times c_{\text{out}} \times c_{\text{in}} \times k^2 \times (\text{mul} + \text{add})$  

$n_{\text{add}} = m^2 \times c_{\text{out}} \times (c_{\text{in}} \times k^2)$  

$n_{\text{sub}} = m^2 \times c_{\text{out}}$  

$n_{\text{mul}} = m^2 \times c_{\text{out}} \times (2 \times c_{\text{in}})$  

$n_{\text{div}} = m^2 \times c_{\text{out}}$  

$n_{\text{root}} = m^2 \times c_{\text{out}}$  

$C_l(l_{\text{Convolutional}}) = [n_{\text{add}}, n_{\text{sub}}, n_{\text{mul}}, n_{\text{div}}, n_{\text{root}}]$.  

B. BOs, 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 Algorithm 2, TOs calculator $T(b, p)$ takes both the BOs extracted by Algorithm 1 and the data type $p$ used in DNN model as inputs. The reason is that at different data precisions, different numbers of TOs are required to do the same operation. $T$ opens the design logic of ALU, analyze the theoretical TOs for each operation module (e.g., 32-bit adder, 16-bit multiplier) with their individual IC designs (the analysis method is introduced in the following sections). 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 return lists of $C_l(l_i), C_{bp}(l_i), C_{bp}(l_i), C_{bp}(l_i)$ due to their similar data format (number of five BOs with fixed order). Suppose $b$ is a piece of BOs information extracted from the target DNN model by $C$, (6) demonstrate how $b$ is processed into TOs with $T$. The calculation of DNN TOs from BOs information could be seen in Algorithm 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 $d_f_{\text{total}},$ $D_{bp}$, $d_{loss}$, and overall TOs $d_{\text{total}}$

$b = [n_{\text{add}}, n_{\text{sub}}, n_{\text{mul}}, n_{\text{div}}, n_{\text{root}}]$

$T(b, p) = T_{\text{add}}(n_{\text{add}}, p) + T_{\text{sub}}(n_{\text{sub}}, p) + T_{\text{mul}}(n_{\text{mul}}, p)$

$ + T_{\text{div}}(n_{\text{div}}, p) + T_{\text{root}}(n_{\text{root}}, p)$

(6)

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 [46] (e.g., adder: ripple-carry and carry look-ahead adder [47]; multiplier: Wallace and Booth–Wallace multiplier [48]). The logic of IC in ALU support different types of BOs, however, not all the BOs have their independent operation unit (e.g., adder is designed for addition, but subtraction is processed in adder by 2s 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.

Algorithm 1: Calculate Layer Based BOs for DNN Model.

Require: DNN model structure $L = \{l_1, l_2, \ldots, l_o\}$

Require: Basic operation calculators: $C_t, C_{bp}, C_{loss}$

Initialise $B_{l_i} = [b_{l_{i1}}, b_{l_{i2}}, \ldots, b_{l_{ip}}]$; $B_{bp} = [b_{bp1}, b_{bp2}, \ldots, b_{bpn}]$

Initialise $b_{loss} = [0, 0, 0, 0, 0] /*$ number of 5 BOs */

for $l_i \in [1, 2, \ldots, o]$ do

$b_{ti} = C_t(l_i)$

$b_{bp_i} = C_{bp}(l_i)$

end for

$b_{loss} = C_{loss}(l_o)$

return $B_{f_i}, B_{bp}$, $b_{loss}$

Fig. 5. Calculation of theoretical TOs based on BOs (Example: multiplication operation on two IEEE 754 FP-32 numbers).
Fig. 6. Method verification (feedforward DNNs on laptop CPU). (a) Feedforward DNN models on laptop: Theoretical TOs and practical energy consumption. (b1) Regression of energy consumption to TOs/FLOPs and residual analysis (Sigmoid). (b2) Regression of energy consumption to TOs/FLOPs and residual analysis (Tanh). (b3) Regression of energy consumption to TOs/FLOPs and residual analysis (GELU). (c) Training of polynomial regression (energy to TOs/FLOPs): Coefficient and performance metrics. (d1) Prediction of DL energy consumption (Sigmoid). (d2) Prediction of DL energy consumption (Tanh). (d3) Prediction of DL energy consumption (GELU). (e) Validation of polynomial regression (energy to TOs/FLOPs): Performance metrics.
Algorithm 2: Calculate Theoretical TOs Based on BOs.

Require: Layer based BOs set
\[ B_t = [b_{t1}, b_{t2}, \ldots, b_{to}]; B_{hp} = [b_{hp1}, b_{hp2}, \ldots, b_{hpo}] \]

Require: BOs for loss calculation \( b_{loss} \)

Require: Data type \( p \)

Require: TOs calculator \( T \)

Initialise
\[ D_t = [d_{t1}, d_{t2}, \ldots, d_{to}]; D_{hp} = [d_{hp1}, d_{hp2}, \ldots, d_{hpo}] \]

Initialise \( d_{total}, d_{hp,\text{total}}, d_{loss}, d_{total} = 0, 0, 0, 0 \)

for \( i \in [0, 1, \ldots, o] \) do
\[ d_{fi} = T(b_{fi}, p) \]
\[ d_{hp,fi} = T(b_{hp,fi}, p) \]
\[ d_{t,\text{total}} + = T(b_{t,i}, p) \]
\[ d_{hp,\text{total}} + = T(b_{hp,i}, p) \]
end for

\( d_{loss} = T(b_{loss}, p) \)
\( d_{total} = d_{t,\text{total}} + d_{hp,\text{total}} + d_{loss} \)

return \( D_t, D_{hp}, d_{t,\text{total}}, d_{hp,\text{total}}, d_{loss}, d_{total} \)

with the Newton–Raphson method [49] by operation units in ALU. Our TOs model in this article uses the basic IC design to calculate the theoretical transistors needed by BOs. Please note, the design of IC is not focused in this article, 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 [50]. And according to the IC design in [18], transistors used for the 64-bit Booth–Wallace multiplier and sweeney, robertson and tocher (SRT) divider are 90 k and 110 k, respectively.

2) \( \text{FP Numbers in TOs Calculation:} \) FP numbers are the most generally used data type in DNNs. As FP used in majority programing language follows IEEE 754 standard, the calculation of theoretical TOs should consider the structure and calculation logic of binary FP with different precision (e.g., FP-32 known as single-precision FP, FP64 as double-precision FP). According to IEEE 754, each FP-32 number contains a 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 an eXclusive OR (XOR) gate (for sign), a 24-bit multiplier/divider (fraction calculation), and an 8-bit adder (exponent calculation). It means the TOs for an FP-32 multiplication is theoretically be calculated by the sum of transistors for a 24-bit multiplier, an 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).

C. TOs Model and Energy Scaling

Theoretical TOs for a given DNN model could be calculated by Algorithms 1 and 2. The process of mapping the scaling of DNN model TOs to its practical energy consumption scaling is demonstrated in Fig. 3(b). First, a list of DNN models with different configurations will be established. Second, 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, 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 following 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 III, 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 [52] (for Intel Architecture CPU [53]), 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 [52]. 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 [54]. However, as our aim is to analyze only calculation energy cost, it is the only way for collecting on-chip component energy data [39], [55] and reported to be fairly accurate by NVIDIA (+−5% error rate [56]) and authors in [57] and [58]. As the problem does not influence theoretical TOs/FLOPs, we will not discuss the accuracy of hardware energy monitoring. During the
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 to maintain the repeatability of experiment) on each model setting with laptop CPU as introduced in Table III, 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 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 $R^2$-square and root mean square error (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 the following equation:

\[
\begin{align*}
\text{DNN on laptop CPU:} & \quad E = 1.71 \times 10^3 + 3.65 \times 10^{-12} \times \text{TOs} \\
\text{DNN on desktop CPU:} & \quad E = 1.73 \times 10^3 + 8.42 \times 10^{-8} \times \text{FLOPs}
\end{align*}
\]

\( B. \) 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 III. 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 \( 80\% / 20\% \). We demonstrate the training and validation of PR energy models with residual analysis in Fig. 8(b) and (c). We also summarize the PR model performance metric (adjusted \( R^2 \)-square and RMSE) in Fig. 8(d) and (e). We also run the same CNN set on a desktop CPU and a laptop CPU separately, please refer to Figs. 2 and 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 laptop processors could be demonstrated by the following equation:

\[
\begin{align*}
\text{CNN on desktop GPU:} & \quad E = 1.76 \times 10^3 + 2.61 \times 10^{-15} \times \text{TOs} \\
\text{CNN on desktop CPU:} & \quad E = 1.81 \times 10^3 + 3.49 \times 10^{-11} \times \text{FLOPs}
\end{align*}
\]

\( C. \) 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 Fig. 6(a) and 8(a) and Figs. 1, 2, and 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 \( R^2 \)-square Error (\( R^2 \)) and RMSE. \( R^2 \) 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 [59]. RMSE represents the differences between values predicted by a model or an estimator and the values observed [60]. From the statistic of \( R^2 \) and RMSE [see Figs. 6(c) and (e) and 8(d) and (e)], both TOs-based and FLOPs-based PR models could accurately fit DNN energy metric and practical energy consumption (\( R^2 \) 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 \( R^2 \) and RMSE than FLOPs-based PR models. This means, TOs are more accurate to be used as a metric for analyzing the scaling law of DNN energy consumption. As we apply PR with 1–3 coefficients to fit the relationship between FLOPs/TOs and practical energy consumption, according to \( R^2 \), 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: \( 2.61 \times 10^{-15} \) Joule; desktop CPU: \( 4.29 \times 10^{-14} \) Joule). Simultaneously, the performance of FLOPs-based and TOs-based PR models for DNN energy estimation in Joule could be summarized in Table IV. 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 [51]) in Fig. 9, from where the energy efficiency of DNN model configurations (number of nodes and the choice of AFs) could be analyzed. From 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 $0.2 \times 10^{14}$ 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 analyzed from DNN structures and configurations. With support to nonlinear operations, TOs are more complex to be calculated than FLOPs/MACs. Compared with methods in [37] and [38], the TOs method is more efficient due to no practical experiment required, but not applicable to black box models.

IV. CONCLUSION

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

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 [61]) 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 multicore energy consumption modeling approaches, we can map the scaling law of DNN energy consumption in multicore environments. We intend to apply this to machine learning techniques used in widespread communication [7] and IoT architectures [3] to have a significant impact.

ACKNOWLEDGMENT

Data are open available and details are in supplementary materials.

REFERENCES

[1] D. Amodei, D. Hernandez, G. Sastry, J. Clark, G. Brockman, and I. Sutskever, “AI and compute.” Heruntergeladen von, 2018. [Online]. Available: https://blog.openai.com/aiand-compute

[2] N. Jones, “The information factories,” Nature, vol. 561, pp. 163–166, 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 Commun. Mag., vol. 54, no. 11, pp. 143–149, Nov. 2016.

[4] J. Wu, S. Guo, J. Li, and D. Zeng, “Big data meet green challenges: Greening big data,” IEEE Syst. J., vol. 10, no. 3, pp. 873–887, Sep. 2016.

[5] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in Proc. Ann. Meeting ACL, 2019, pp. 3645–3650.

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

[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 Veh. Technol. Mag., vol. 16, no. 1, pp. 29–39, Mar. 2021.

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

[9] T. Brown et al., “Language models are few-shot learners,” Adv. Neural Inf. Process. Syst., vol. 33, pp. 1877–1901, 2020.

[10] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, “Green AI,” Commun. 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 Spectr., vol. 59, no. 4, pp. 22–50, Apr. 2022.

[12] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, “Energy efficient federated learning over wireless communication networks,” IEEE Trans. Wireless Commun., vol. 20, no. 3, pp. 1935–1949, Mar. 2020.

[13] B. Li et al., “Random sketch learning for deep neural networks in edge computing,” Nature Comput. Sci., 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,” 2016, arXiv:1611.06440.
