Neural Networks 178 (2024) 106419

Contents lists available at ScienceDirect

Neural Networks

journal homepage: www.elsevier.com/locate/neunet

Full Length Article

On energy complexity of fully-connected layers

Jiří Šíma [a,*], Jérémie Cabessa [b], Petra Vidnerová [a]

[a] Institute of Computer Science of the Czech Academy of Sciences, Pod Vodárenskou věží 271/2, Prague 8, 182 00, Czechia
[b] DAVID Laboratory, University of Versailles Saint-Quentin (UVSQ), University Paris-Saclay, 45 avenue des États-Unis, Versailles, 78035, France

ARTICLE INFO

Keywords:
Deep neural networks
Convolutional neural networks
Fully-connected layer
Energy complexity
Energy consumption
Dataflow

ABSTRACT

The massive increase in the size of deep neural networks (DNNs) is accompanied by a significant increase in energy consumption of their hardware implementations which is critical for their widespread deployment in low-power mobile devices. In our previous work, an abstract hardware-independent model of energy complexity for convolutional neural networks (CNNs) has been proposed and experimentally validated. Based on this model, we provide a theoretical analysis of energy complexity related to the computation of a fully-connected layer when its inputs, outputs, and weights are transferred between two kinds of memories (DRAM and Buffer). First, we establish a general lower bound on this energy complexity. Then, we present two dataflows and calculate their energy costs to achieve the corresponding upper bounds. In the case of a partitioned Buffer, we prove by the weak duality theorem from linear programming that the lower and upper bounds coincide up to an additive constant, and therefore establish the optimal energy complexity. Finally, the asymptotically optimal quadratic energy complexity of fully-connected layers is experimentally validated by estimating their energy consumption on the Simba and Eyeriss hardware.

1. Energy complexity model for CNNs

Deep neural networks (DNNs) represent a cutting-edge machine learning technology, with countless applications in computer vision, natural language processing (NLP), speech recognition, robotics, etc. In particular, the transformer models have revolutionized the world of NLP (Vaswani et al., 2017), and further led to the development of large language models like GPT (Brown et al., 2020), PaLM (Chowdhery et al., 2023), and LLaMA (Touvron et al., 2023). The transformer models have also been extended into the field of computer vision (ViT) (Dosovitskiy et al., 2021) and tasks based on tabular data (Tab-Transformer) (Huang et al., 2020). However, their tremendous performance is achieved at the cost of a huge number of parameters. For instance, GPT-3 contains 175B (billion) parameters and necessitated 34 days of training on 1024 GPUs consuming 4.68×10^{12} joules of energy (Luccioni et al., 2022). Similarly, PaLM has 540B parameters and LLaMA's size ranges from 7B to 65B parameters.

Thus, deep learning models are very computationally demanding and consume an enormous amount of energy, which can be critical to their deployment in practical applications. For example, an increasing number of embedded (edge) devices rely on DNNs to deliver sophisticated services, such as autonomous surveillance systems utilizing advanced object recognition, personal assistants employing machine translation, smart healthcare applications, and more (Lyu et al., 2023; Mishra et al., 2020). However, with the ever-growing use of mobile devices, such as smartphones, smartwatches, or smartglasses, comes the issue of the implementation, deployment, and portability of an already trained DNN on low-power hardware operated on batteries, which is a major bottleneck to the development of smart wearable electronics. Therefore, recent research has focused on developing methods that enable energy-efficient processing of DNNs (Sze et al., 2017, 2020).

There are basically two main approaches to reduce the energy cost of DNNs. The first approach is suitable for error-tolerant applications such as image classification where enormous amount of energy can be saved at the cost of only a small loss in accuracy by using *approximate computing* methods (Armeniakos et al., 2023; Deng et al., 2020; Li et al., 2023; Lyu et al., 2023; Mittal, 2016; Tang et al., 2024), e.g. low float precision (Gupta et al., 2015), approximate multipliers (Ansari et al., 2020), etc. In the second approach the computational cost is reduced through *hardware design* (Jouppi et al., 2018; Silvano et al., 2023) including massive parallelism where DNNs are implemented on a variety of hardware platforms such as GPUs (Zhou et al., 2018), FPGAs (Mittal, 2020), in-memory computing architectures (Mittal et al., 2021), etc.

For a specific DNN hardware implementation, the real energy consumption of the inference process can be either practically measured or analytically estimated using physical laws. This energy consumption depends on parameters and constants related to the hardware architecture, and hence, its evaluation varies for different DNN hardware

* Corresponding author.
E-mail addresses: sima@cs.cas.cz (J. Šíma), jeremie.cabessa@uvsq.fr (J. Cabessa), petra@cs.cas.cz (P. Vidnerová).

https://doi.org/10.1016/j.neunet.2024.106419
Received 21 November 2023; Received in revised form 16 May 2024; Accepted 29 May 2024
Available online 31 May 2024

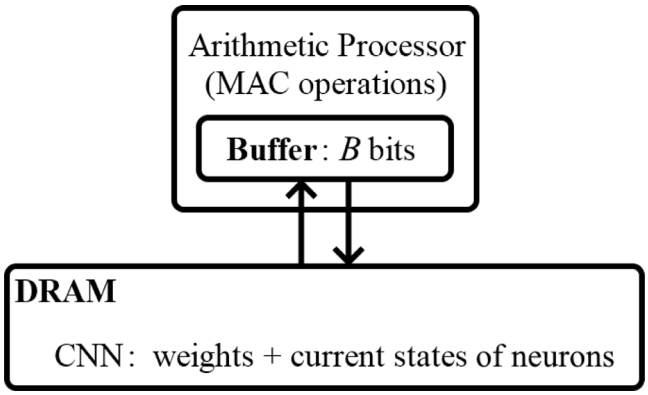

Fig. 1. The energy complexity model.

implementations. Some software tools such as Accelergy (Wu et al., 2019) and Timeloop (Parashar et al., 2019) can calculate and optimize, respectively, the energy consumption of a particular DNN on various hardware platforms including the Simba (Shao et al., 2019) and Eyeriss (Chen et al., 2016) architectures.

It has been empirically observed that the energy cost of DNN processing mainly consists of two components: the computation energy, and the data energy which represents around 70% of the total cost (Yang et al., 2017). The *computation energy* is needed for performing arithmetic operations, especially the so-called multiply-and-accumulate (MAC) operations ($S \leftarrow S + wx$ on floats S, w, x), used to compute the weighted sums of inputs in neurons. The *data energy* is required for moving the data inside the memory hierarchy of the hardware (dataflow), and is related to the number of memory accesses.

In the general context of high-performance computing, heterogeneous architectures merging two kinds of memories, CPUs and GPUs, are considered. The task scheduling problem aims at minimizing the processing time — and thus the energy consumption — of a set of tasks involving various types of data (see Gonthier et al., 2023, and the references therein). This optimization is achieved through three objectives: minimizing data transfers throughout the memories, ensuring overlap between data transfers and task computations, and optimizing the eviction of previously-loaded data. In this context, the particular problem, close to ours, of scheduling a set of tasks on one GPU with limited memory, where the tasks share some of their input data but are otherwise independent, is shown to be NP-complete and is in turn addressed by means of different heuristics (Gonthier et al., 2023).

Along these lines, we propose a theoretical study of the energy complexity of DNNs where the computational process involves CPU/GPU-like data transfers. In a recent paper (Šíma et al., 2024), we have introduced a simplified machine-independent model of energy complexity for convolutional neural networks (CNNs). This model abstracts from the implementation details related to different hardware platforms, and preserves the asymptotic energy complexity of the CNN inference. It is composed of only two memory levels called *DRAM* and *Buffer*, as illustrated in Fig. 1. The network parameters and states are stored in DRAM, and the arithmetic operations are performed only over numerical data stored in Buffer which has a constant capacity of B bits. The transfer of data between the two memories determines the dataflow. We assume that any floating-point number is transferred as a separate, indivisible, and uncompressed block of b bits.

The main idea behind this model is that, for a given CNN stored in DRAM, the three arguments (input x, weight w and accumulated output S) of any MAC operation ($S \leftarrow S + wx$) employed for the evaluation of the network must occur together at the same time in Buffer. This requirement is common to all conceivable hardware implementations of CNNs, making the model universal. The CNN inference thus requires a certain number of data transfers between DRAM and Buffer (i.e., the number of DRAM accesses multiplied by the number of b bits in a float number), which corresponds to our measure of the data energy.

For simplicity, we assume that the energy cost is not optimized across multiple CNN layers (as, e.g., Alwani et al., 2016, for instance). Hence, the energy complexity is defined as a simple sum over only separate convolutional layers including the fully-connected ones as a special case, while the less energy-intensive max pooling layers are omitted. Formally,

$$E = \sum_{\text{convolutional layer } \lambda} \left(E^{\lambda}_{\text{comp}} + E^{\lambda}_{\text{data}} \right) \tag{1}$$

where the computation energy $E^{\lambda}_{\text{comp}}$ and the data energy $E^{\lambda}_{\text{data}}$ for evaluating a convolutional layer λ is proportional to the corresponding number of MACs and DRAM accesses, respectively.

The energy complexity model of CNNs has been exploited for calculating the theoretical energy of processing convolutional layers in the context of two common dataflows with write-once outputs and read-once inputs, respectively, and read-once weights, under realistic buffer capacity constraints (Šíma et al., 2024). These dataflows provide upper bounds on the energy complexity of CNN layers, which have been compared to the real energy consumptions estimated for the Simba (Shao et al., 2019) and Eyeriss (Chen et al., 2016) architectures by using the Timeloop/Accelergy software tool (Parashar et al., 2019; Wu et al., 2019).

It turns out that the theoretical upper bounds fit asymptotically very well the empirical optimal energy consumptions, when individual parameters of a convolutional layer such as the height, width, depth, kernel size, and stride are varied. This was validated by statistical linearity and quadraticity tests (Šíma et al., 2024). Thus, the introduced energy complexity model appears to be able to asymptotically capture all important sources of energy consumption that are common to different CNN hardware implementations. The model can also be exploited for proving lower bounds on energy complexity of CNNs, in order to establish asymptotic limits on energy efficiency of any CNN hardware accelerators.

In this paper, we investigate energy complexity of *fully-connected layers* which can be viewed as special convolutional layers where feature maps are reduced to single neurons. First, we derive a general lower bound on the data energy complexity. Then we present two types of dataflows in which each weight and each output (or alternatively each input) are read into Buffer once only. In the first dataflow, the Buffer memory is assumed to be partitioned into two separate parts of given fixed capacities for inputs and outputs, respectively. The second dataflow is parameterized by the maximum number of inputs residing in Buffer at the same time. We determine the data energy of both dataflows, which provides upper bounds on energy complexity. Moreover, for the first dataflow, we prove that the lower and upper bounds coincide up to an additive constant, by means of the weak duality theorem from linear programming. The optimal energy complexity for fully-connected layers in situations where Buffer is partitioned into two separate parts for inputs and outputs, respectively, ensues.

The presented upper bounds differ only by a linear additive term from the derived lower bound, which provides the asymptotically optimal quadratic data energy complexity of evaluating a fully-connected layer in terms of the number of its inputs and outputs. This theoretical energy complexity is also compared to the real energy consumptions estimated for the Simba and Eyeriss hardware architectures by the Timeloop/Accelergy tool. It turns out that it matches very well when the numbers of inputs, outputs, and weights of fully-connected layers are varied separately, which is validated by the statistical linearity tests.

The paper is organized as follows. Section 2 formally defines energy complexity for fully-connected layers. A general lower bound on this energy is derived in Section 3. Section 4 presents two dataflows with their associated upper bounds on the energy. In Section 5, the matching and thus optimal lower bound is derived for the case of

a partitioned Buffer. Section 6 experimentally validates the asymptotically optimal quadratic energy complexity of fully-connected layers. Section 7 summarizes the results and discusses open problems. A preliminary conference version (Šíma & Cabessa, 2023) of this paper is substantially expanded here to include a new general lower bound on energy, a detailed description of dataflows, and experimental validation of energy complexity.

2. Energy complexity of fully-connected layers

Consider a deep (e.g. convolutional) neural network of depth D and a layer index λ of some of its fully-connected layers, where $0 < \lambda \leq D$ (note that the index $\lambda = 0$ is reserved for the input layer). We assume that the λ-th layer, referred to as layer λ, is composed of m neurons (units) $y_1, \ldots, y_m$, each of which is receiving real-weighted connections from the n neurons $x_1, \ldots, x_n$ in the previous layer $\lambda - 1$.

This situation can be viewed as a complete weighted bipartite graph $G = (X, Y, E, w)$ where $X = \{x_1, \ldots, x_n\}$ and $Y = \{y_1, \ldots, y_m\}$ are disjoint sets of *inputs* and *outputs*, respectively, $E = X \times Y$ is the set of directed edges between inputs and outputs, and $w : X \times Y \to \mathbb{R}$ is a function that associates each edge $(x_i, y_j) \in E$ with a real *weight* w_{ji}, for every $j \in \{1, \ldots, m\}$ and every $i \in \{1, \ldots, n\}$. Moreover, each output y_j is associated with a real *bias* w_{j0}, for every $j \in \{1, \ldots, m\}$. In the sequel, the symbols x_i and y_j will be indifferently used to denote input and output units as well as numerical values held by them, respectively. The distinction will be clear from the context.

The *computation of layer* λ refers to the computation of the output values $y_1, \ldots, y_m$ based on the input values $x_1, \ldots, x_n$, the weights and biases w_{ji}, for $j \in \{1, \ldots, m\}$ and $i \in \{0, \ldots, n\}$, which is controlled by the following equations:

$$y_j = \sigma\left(w_{j0} + \sum_{i=1}^{n} w_{ji} x_i\right) \quad \text{for every } j = 1, \ldots, m, \tag{2}$$

where σ is the *activation function*. Typically σ can be taken as the *rectified linear unit* activation function given by $\mathrm{ReLU}(x) = \max(0, x)$.

The computation energy $E^{\lambda}_{\mathrm{comp}}$ in (1) required for the computation of layer λ can be evaluated directly. According to (2), each output y_j requires one initialization step followed by n MAC updates:

$$S \leftarrow w_{j0} \quad \text{and} \quad S \leftarrow S + w_{ji} x_i \quad \text{for } i = 1, \ldots, n,$$

where the current value of S is referred to as the *accumulated output* y_j. Hence, the total number of MAC operations needed for computing the outputs $y_1, \ldots, y_m$ in (2) is mn. The computation energy is thus given by

$$E^{\lambda}_{\mathrm{comp}} = C_b\, mn \tag{3}$$

where the constant C_b is the energy cost to perform one MAC operation over b-bit floating-point numbers on a given hardware architecture. Note that the dependence of C_b on b is non-uniform because the design of the MAC circuit inside a microprocessor differs for each b. For example, $C_8 = 0.56\,\mathrm{pJ}$ and $C_{16} = 2.20\,\mathrm{pJ}$ was estimated by the Timeloop/Accelergy software tool (Parashar et al., 2019; Wu et al., 2019) for the 8-bit Simba (Shao et al., 2019) and 16-bit Eyeriss (Chen et al., 2016) hardware architectures, respectively. Nevertheless, in our machine-independent energy complexity model, concrete values of constant C_b do not play a role in the asymptotic bound $E^{\lambda}_{\mathrm{comp}} = \Theta(mn)$ on the computation energy which is clearly optimal since each of the mn weights occurs in a different MAC operation.

We now focus on the data energy $E^{\lambda}_{\mathrm{data}}$ in (1) necessary for the computation of layer λ. This energy cost can be split into three components that count the DRAM accesses for the outputs, inputs, and weights separately:

$$E^{\lambda}_{\mathrm{data}} = E^{\lambda}_{\mathrm{outputs}} + E^{\lambda}_{\mathrm{inputs}} + E^{\lambda}_{\mathrm{weights}}\,. \tag{4}$$

In order to evaluate the sums in (2), each pair of input and accumulated output (x_i, y_j) must occur in Buffer at least once. For this purpose, each input x_i and output y_j needs to be read from DRAM at least once. Furthermore, each output y_j must also be written back to DRAM sometime after its reading in order to store its current value. By contrast, each weight w_{ji} only needs to occur in Buffer once when the associated pair (x_i, y_j) meets in Buffer for the first time. It follows that each weight w_{ji} requires only one reading from DRAM, which in turn amounts to mn DRAM accesses for reading all the weights.

Let ν and μ be the numbers of DRAM accesses to read inputs and outputs (or biases when initialized), respectively, and b be the number of bits in the floating point representation of outputs, inputs, and weights. The data energy (4) can thus be written as

$$E^{\lambda}_{\mathrm{data}} = b\,(2\mu + \nu + mn)\,, \tag{5}$$

since each output is read from and later written back to DRAM, which corresponds to two DRAM accesses, as opposed to the inputs and weights which are only read from DRAM. Note that in our machine-independent energy complexity model, the data energy $E^{\lambda}_{\mathrm{data}}$ is defined in (5) simply as the number of transferred bits between DRAM and Buffer, each with the same unit energy cost in joules, which is in contrast to $E^{\lambda}_{\mathrm{comp}}$ in (3) that includes the actual comparable energy costs C_b of MAC circuits. This is because the bit transfers can have different meanings even within a single hardware implementation, let alone in distinct hardware architectures with different memory hierarchies. Consequently, in order to optimize the data energy (4), it is sufficient to minimize the quantity $2\mu + \nu$.

3. A lower bound on energy complexity

We will now derive a general lower bound on the data energy (4) for fully-connected layers. Assume that Buffer has a constant size of $B = b(\beta + 1)$ bits, where $\beta > 1$ floats are reserved for storing inputs and outputs, and the remaining capacity of one float is dedicated to weights. For notational simplicity, assume that $\beta - 1$ divides m, $\beta > 2$, and $m \leq n$, while the proof for the other cases is analogous. Note that, by reading a single input or output into Buffer, one can get at most $\beta - 1$ new input–output pairs in Buffer.

In a dataflow, let r_1 be the maximum number of times one input, being read into Buffer, creates exactly $\beta - 1$ *new* input–output pairs. Denote by $x^* \in X$ one of such inputs and let $Z_j \subset Y$ be the sets of outputs forming the respective $\beta - 1 = |Z_j|$ new pairs $\{x^*\} \times Z_j$ for every $j \in \{1, \ldots, r_1\}$. Analogously, let r_2 be the maximum number of times one output, being read into Buffer, produces exactly $\beta - 1$ new pairs. Denote by $y^* \in Y$ one of such outputs and let $Z_j \subset X$ be the sets of inputs creating the respective $\beta - 1 = |Z_j|$ new pairs $Z_j \times \{y^*\}$ for every $j \in \{r_1 + 1, r_1 + 2, \ldots, r_1 + r_2\}$. Hereafter, we will focus on the sets of outputs, Z_j for $j \in \{1, \ldots, r_1\}$, while the analysis for the sets of inputs, Z_j for $j \in \{r_1 + 1, \ldots, r_1 + r_2\}$, is analogous. Note that the sets Z_j are pairwise disjoint for all $j \in \{1, \ldots, r_1\}$, because they yield the input–output pairs with x^* that are new along the dataflow.

For each $j \in \{1, \ldots, r_1\}$, we denote by α_j a DRAM access through which an input $x^*_j \in X$ is read into Buffer that already contains outputs from Z_j, which generates exactly $\beta - 1$ new pairs $\{x^*_j\} \times Z_j$, while the immediately preceding reading into Buffer produces less than $\beta - 1$ new pairs. From the definition of x^*, there is at least one such DRAM access for each $j \in \{1, \ldots, r_1\}$, and we choose any of them as α_j if there are more.

For each $j \in \{1, \ldots, r_1\}$, we define $\beta - 1$ DRAM accesses ζ_{ji}, indexed by $i \in \{1, \ldots, \beta - 1\}$ according to the time order along the dataflow, through which the $\beta - 1$ outputs in Z_j are read into Buffer, each one last before the DRAM access α_j. Observe that ζ_{ji} are pairwise distinct for all $j \in \{1, \ldots, r_1\}$ and $i \in \{1, \ldots, \beta - 1\}$ because the sets Z_j are pairwise disjoint. For every $i \in \{1, \ldots, \beta - 1\}$, denote by $y_{ji} \in Z_j$ the output that is read through the DRAM access ζ_{ji}, and let m_{ji} be the number of new input–output pairs in Buffer generated through ζ_{ji}.

For every $i \in \{1, \ldots, \beta - 1\}$, there are i outputs $y_{j1}, \ldots, y_{ji} \in Z_j$ in Buffer after the DRAM access ζ_{ji} which remain there at least until the

DRAM access α_j. In order to fit the Buffer capacity β, there are thus at most $\beta - i$ inputs in Buffer after ζ_{ji}, which implies $m_{ji} \leq \beta - i$ for every $i \in \{1, \ldots, \beta - 1\}$. Let $k_j \in \{1, \ldots, \beta - 1\}$ be the maximum number of new input–output pairs in Buffer that are produced by a DRAM access ζ_{ji} over $i \in \{1, \ldots, \beta - 1\}$. It follows that

$$m_{ji} \leq \begin{cases} k_j & \text{for } 1 \leq i \leq \beta - k_j \\ \beta - i & \text{for } \beta - k_j + 1 \leq i \leq \beta - 1 \,. \end{cases} \tag{6}$$

Altogether, the number of new input–output pairs in Buffer generated through the DRAM accesses ζ_{ji} for all $i \in \{1, \ldots, \beta - 1\}$, can be upper bounded as

$$\begin{aligned} \sum_{i=1}^{\beta-1} m_{ji} &\leq \sum_{i=1}^{\beta-k_j} k_j + \sum_{i=\beta-k_j+1}^{\beta-1} (\beta - i) = (\beta - k_j)k_j + \sum_{i=1}^{k_j-1} (k_j - i) \\ &= (\beta - 1)k_j - \sum_{i=1}^{k_j-1} i = (\beta - 1)k_j - \frac{k_j(k_j - 1)}{2} \end{aligned} \tag{7}$$

according to (6).

For each $j \in \{1, \ldots, r_1\}$, we thus have $\beta - 1$ unique DRAM accesses $\zeta_{j1}, \ldots, \zeta_{j,\beta-1}$ through which the outputs $y_{j1}, \ldots, y_{j,\beta-1} \in Z_j$ are read, respectively, creating at most $(\beta-1)k_j - k_j(k_j-1)/2$ new input–output pairs in Buffer, according to (7). Analogously, for each $j \in \{r_1+1, \ldots, r_1+r_2\}$, we have $\beta - 1$ unique DRAM accesses $\zeta_{j,1}, \ldots, \zeta_{j,\beta-1}$ through which the inputs from Z_j are read that yield at most $(\beta - 1)k_j - k_j(k_j - 1)/2$ new input–output pairs in Buffer. Let $R \subseteq \{j \mid 1 \leq j \leq r_1 + r_2\}$ be a subset of indices of

$$r = \min\left(\frac{m}{\beta - 1}, r_1 + r_2\right) \tag{8}$$

largest k_j, which means $k_j \geq k_\ell$ for every $j \in R$ and $\ell \in \{1, \ldots, r_1 + r_2\} \setminus R$. The number of new input–output couples generated through the $(\beta - 1)r$ DRAM accesses $\zeta_{j1}, \ldots, \zeta_{j,\beta-1}$ for all $j \in R$, can be upper bounded as

$$\sum_{j \in R} \sum_{i=1}^{\beta-1} m_{ji} \leq (\beta - 1) \sum_{j \in R} k_j - \sum_{j \in R} \frac{k_j^2 - k_j}{2} \tag{9}$$

according to (7).

Let s be the number of DRAM read accesses that produce exactly $\beta - 1$ new pairs, excluding $\zeta_{j1}, \ldots, \zeta_{j,\beta-1}$ for all $j \in R$. By the definition of r_1 and r_2, we have

$$s \leq \sum_{j \in R} (n - k_j) \tag{10}$$

because $s \leq mn/(\beta - 1)$ due to the fact that the total number of input–output pairs is mn, and there are at most $n - k_j$ inputs or $m - k_j \leq n - k_j$ outputs that can create new $\beta - 1$ input–output pairs with the outputs or inputs from Z_j, respectively, for all $j \in R$. Since the maximum s, determined in (10), minimizes the number of DRAM read accesses, any dataflow satisfies

$$\mu + \nu \geq (\beta - 1)r + \sum_{j \in R} (n - k_j) + q + 1 \tag{11}$$

where $q + 1$ denotes the number of remaining DRAM read accesses that produce less than $\beta - 1$ new pairs, including the very first DRAM access yielding no pair and excluding the $(\beta - 1)r$ DRAM accesses $\zeta_{j1}, \ldots, \zeta_{j,\beta-1}$ for all $j \in R$.

Since all the mn input–output pairs have to occur in Buffer, we have

$$(\beta - 1) \sum_{j \in R} k_j - \sum_{j \in R} \frac{k_j^2 - k_j}{2} + (\beta - 1) \sum_{j \in R} (n - k_j) + (\beta - 2)q \geq mn \tag{12}$$

according to (9) and (10), because each of the remaining q DRAM read accesses yields at most $\beta - 2$ new pairs. This reduces to

$$q \geq \frac{mn}{\beta - 2} - \frac{(\beta - 1)rn}{\beta - 2} + \sum_{j \in R} \frac{k_j^2 - k_j}{2(\beta - 2)} \tag{13}$$

which, plugged into (11), gives

$$\mu + \nu \geq \left(\beta - 1 - \frac{n}{\beta - 2}\right) r + \frac{mn}{\beta - 2} + \sum_{j \in R} \frac{k_j^2 - (2\beta - 3)k_j}{2(\beta - 2)} + 1 \,. \tag{14}$$

Each term of the summation in (14) meets

$$\frac{k_j^2 - (2\beta - 3)k_j}{2(\beta - 2)} \geq -\frac{\beta - 1}{2} \tag{15}$$

for $j \in R$, because the inequality (15) is equivalent to

$$(k_j - (\beta - 1))(k_j - (\beta - 2)) \geq 0 \tag{16}$$

which holds for every integer $k_j \in \{1, \ldots, \beta - 1\}$. Hence, the sum in (14) can be lower bounded by $-(\beta - 1)r/2$ as

$$\mu + \nu \geq \left(\frac{\beta - 1}{2} - \frac{n}{\beta - 2}\right) r + \frac{mn}{\beta - 2} + 1 \tag{17}$$

which is a linear function in terms of r, whose slope is negative for sufficiently large $n > (\beta - 1)(\beta - 2)/2$. Hence, the lower bound (17) remains valid after we substitute the maximum feasible value for r, that is, $r = \frac{m}{\beta - 1}$ according to (8), which gives

$$\mu + \nu \geq \frac{mn}{\beta - 1} + \frac{m}{2} + 1 = \frac{m(n - 1)}{\beta - 1} + \left(\frac{\beta + 1}{\beta - 1}\right) \frac{m}{2} + 1 \tag{18}$$

for $n > (\beta - 1)(\beta - 2)/2$.

Since the biases of all m outputs must first be read into Buffer, we have $\mu \geq m$, and thus,

$$2\mu + \nu \geq \frac{m(n - 1)}{\beta - 1} + \frac{3\beta - 1}{2(\beta - 1)} m + 1 \,. \tag{19}$$

This provides the general lower bound on the data energy of a fully-connected layer λ:

$$E_{\text{data}}^{\lambda} \geq b \left(mn + \frac{m(n - 1)}{\beta - 1} + \frac{3}{2} m + 1\right) \tag{20}$$

according to (5).

4. Upper bounds on energy complexity

Any correct dataflow for processing a fully-connected layer can be described by a sequence of p sets $B_0, B_1, \ldots, B_p \subseteq X \cup Y$, each of which being composed of vertices in G, that represent the successive contents of Buffer (excluding weights) after each DRAM access to read an input or output, in the course of evaluating the sums in (2). The sequence satisfies the following conditions:

1. $B_0 = \emptyset$
2. $|B_t| \leq \beta$ for every $t = 1, \ldots, p$
3. $|B_t \setminus B_{t-1}| = 1$ and $|B_{t-1} \setminus B_t| \leq 1$ for every $t = 1, \ldots, p$
4. $Y \subseteq \bigcup \{B_t \mid x \in B_t \text{ and } 1 \leq t \leq p\}$ for every $x \in X$

and its length p is the total number of DRAM read accesses,

$$p = \mu + \nu \,. \tag{21}$$

Condition 1 assumes empty Buffer at the beginning, and condition 2 guarantees that its size is not exceeded. Condition 3 ensures that, by reading a single input or output into Buffer, at most one input or output is overwritten. Condition 4 ensures that all of the outputs meet every input in Buffer.

In the two following subsections, we present two dataflows for a fixed and bounded number of inputs in Buffer, respectively, such that each output is read into Buffer only once (i.e., when initialized by a corresponding bias), which means that

$$\mu = m \,. \tag{22}$$

Clearly, the role of inputs and outputs can be reversed in these dataflows.

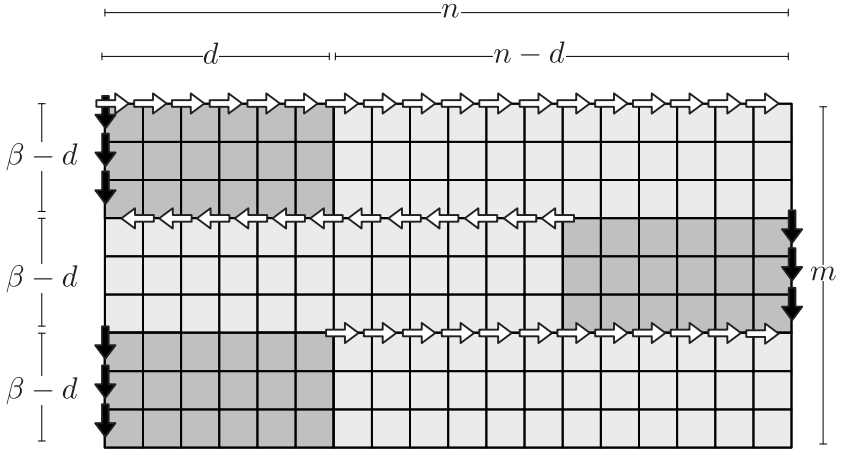

Fig. 2. Illustration of the dataflow for a partitioned Buffer with d inputs and $\beta-d$ outputs. The column and row indices represent inputs $x_1,\dots,x_n$ and outputs $y_1,\dots,y_m$, respectively. The horizontal (white) and vertical (black) arrows represent input and output readings into Buffer, respectively. Every time a new input–output pair (x_i, y_j) meets in Buffer, the weight w_{ji} is read and the accumulated output y_j is updated by the MAC operation $y_j \leftarrow y_j + w_{ji}x_i$. At the beginning, the first d inputs are read (6 first top horizontal arrows). Then, the first block of $\beta-d$ outputs is read (top vertical arrows), which leads to the meeting of new input–output pairs (top left cells, dark region). Then, the remaining $n-d$ inputs are read (remaining top horizontal arrows), leading to new input–output pairs (top right cells, light region). At this point, Buffer contains the d inputs that were lastly read and the second block of $\beta-d$ outputs is read (middle vertical arrows), which yields new input–output pairs (middle right cells, dark region). Afterwards, the remaining $n-d$ inputs are read in the backward direction (middle horizontal arrows), generating new input–output pairs (middle left cells, light region). The dataflow continues in this way by reading outputs and inputs alternatively.

4.1. Fixed number of inputs in Buffer

For the first dataflow, we assume that Buffer is partitioned into two separate parts for inputs and outputs, respectively, and contains one more float for reading the weights. One part is reserved for storing d inputs and the second one to store $\beta-d$ outputs, where d is a fixed parameter such that $1 \le d \le \beta-1$. For notational simplicity, we assume that $\beta-d$ divides m.

The main idea of this dataflow is that the m outputs are split into $\frac{m}{\beta-d}$ groups. These groups, each of size $\beta-d$ outputs, are read into Buffer one after the other with the next group overwriting the current one at specific times when Buffer already contains d inputs. For each such group loaded into Buffer, all the remaining $n-d$ inputs are read into Buffer one by one in such a way that the currently read input replaces a previously read one. This procedure ensures that all the mn input–output pairs will occur in Buffer within its capacity of d inputs and $\beta-d$ outputs. This dataflow is illustrated in Fig. 2.

The dataflow is formally described in Algorithm 1 where the comments (beginning with double slashes) specify the current Buffer contents $B_t \subseteq X \cup Y$ after t DRAM read accesses. Thus, the sequence of sets $B_0, B_1, \dots, B_p$ meets conditions 1–4, $|B_t \cap X| \le d$, and $|B_t \cap Y| \le \beta-d$ for every $t = 0,\dots,p$.

At the beginning when Buffer is empty (line 1), the first d inputs are read into Buffer (loop 2–4) so that $B_d = \{x_1,\dots,x_d\}$ (line 4). Then the algorithm continues with the outer *for* loop 5–27 which goes through all the $\frac{m}{\beta-d}$ groups of $\beta-d$ outputs, indexed as $k = 0,\dots,\frac{m}{\beta-d}-1$. These $\beta-d$ outputs are read into Buffer during the first inner loop 6–13. In particular, for the first group of outputs with the index $k=0$ (line 7) when Buffer contains only the d inputs $x_1,\dots,x_d$, these $\beta-d$ outputs $y_1,\dots,y_{\beta-d}$ are just read into Buffer (line 8) in which there is enough space for them. This means $B_\beta = \{x_1,\dots,x_d,y_1,\dots,y_{\beta-d}\}$ (cf. line 13 for $k=0$).

For the next group of outputs with the index $k>0$ (line 9), these $\beta-d$ outputs $y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}$ are read into Buffer one by one replacing the $\beta-d$ outputs $y_{(k-1)(\beta-d)+1},\dots,y_{k(\beta-d)}$ from the previous group with the index $k-1$ (lines 10–11). Thus, the whole group of outputs with the index k is then contained in Buffer (line 13 where the index of $B_{k(n+\beta-2d)+\beta}$ for $k>0$ takes into account also the DRAM accesses through which inputs are read into Buffer in between the readings of the two groups, as described on lines 14–26 and commented below).

Algorithm 1 The dataflow with a fixed number d of inputs in Buffer.

1: // $B_0 = \emptyset$
2: **for** $i = 1$ **to** d **do**
3: **read** x_i into `Buffer` // $B_i = \{x_1,\dots,x_i\}$
4: **end for** // $B_d = \{x_1,\dots,x_d\}$
5: **for** $k = 0$ **to** $\frac{m}{\beta-d}-1$ **do**
6: **for** $j = 1$ **to** $\beta-d$ **do**
7: **if** $k = 0$ **then**
8: **read** y_j into `Buffer` // $B_{d+j} = \{x_1,\dots,x_d,y_1,\dots,y_j\}$
9: **else**
10: **read** $y_{k(\beta-d)+j}$ into `Buffer` by overwriting $y_{(k-1)(\beta-d)+j}$
11: // $\{y_{k(\beta-d)+1},\dots,y_{k(\beta-d)+j},y_{(k-1)(\beta-d)+j+1},\dots,y_{k(\beta-d)}\} \subset B_{k(n+\beta-2d)+d+j}$
12: **end if**
13: **end for** // $\{y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\} \subset B_{k(n+\beta-2d)+\beta}$
14: **if** k is even **then**
15: // $B_{k(n+\beta-2d)+\beta} = \{x_1,\dots,x_d,y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
16: **for** $i = 1$ **to** $n-d$ **do**
17: **read** x_{i+d} into `Buffer` by overwriting x_i
18: // $B_{k(n+\beta-2d)+\beta+i} = \{x_{i+1},\dots,x_{i+d},y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
19: **end for** // $B_{(k+1)(n+\beta-2d)+d} = \{x_{n-d+1},\dots,x_n,y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
20: **else**
21: // $B_{k(n+\beta-2d)+\beta} = \{x_{n-d+1},\dots,x_n,y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
22: **for** $i = n-d$ **downto** 1 **do**
23: **read** x_i into `Buffer` by overwriting x_{i+d}
24: // $B_{k(n+\beta-2d)+n+\beta-d-i+1} = \{x_i,\dots,x_{i+d-1},y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
25: **end for** // $B_{(k+1)(n+\beta-2d)+d} = \{x_1,\dots,x_d,y_{k(\beta-d)+1},\dots,y_{(k+1)(\beta-d)}\}$
26: **end if**
27: **end for**

The following second inner *for* loop is used to read the $n-d$ inputs into Buffer one by one in addition to the d inputs that are already in Buffer. In order to keep the capacity of d inputs in Buffer, each newly read input rewrites an input that has resided in Buffer for the longest time. Namely, there are two alternating versions of this loop, depending on whether k is even or not (line 14). For k even, the loop 16–19 starts with Buffer including the d inputs $x_1,\dots,x_d$ (line 15), reads the inputs forward (line 17), and finishes with the d inputs $x_{n-d+1},\dots,x_n$ in Buffer (line 19). On the contrary, for k odd (line 20), the loop 22–25 starts with Buffer including the d inputs $x_{n-d+1},\dots,x_n$ (line 21), reads the

inputs backward (line 23), and finishes with the d inputs $x_1, \dots, x_d$ in Buffer (line 25). In both cases, all the n inputs meet each of the $\beta - d$ outputs of the group with the index k which resides currently in Buffer. This is repeated for every group of outputs (outer loop 5–27), which guarantees that all the mn input–output pairs will occur in Buffer.

We will calculate the number p of DRAM read accesses in the dataflow described by Algorithm 1. After the first d inputs are read into Buffer in the loop 2–4, the outer loop 5–27 which runs $\frac{m}{\beta-d}$ times, includes $\beta - d$ DRAM accesses to read outputs in the loop 6–13 and $n - d$ DRAM accesses for reading inputs either in the loop 16–19 or in the loop 22–25. Altogether, we have

$$p = d + \frac{m}{\beta - d}\left((\beta - d) + (n - d)\right) = \frac{m(n-d)}{\beta - d} + m + d\,. \tag{23}$$

Hence, this dataflow provides an upper bound on the data energy of a fully-connected layer λ:

$$E^{\lambda}_{\text{data}} \le b\left(mn + \frac{m(n-d)}{\beta - d} + 2m + d\right) \tag{24}$$

according to (5), (21), and (22). This upper bound takes the smallest value for $d = 1$, provided that $n \ge \beta$, since $n \ge \beta$ is equivalent to

$$\frac{m(n-1)}{\beta - 1} \le \frac{m(n-d)}{\beta - d}.$$

Furthermore, an alternative upper bound to (24) is obtained when the roles of the inputs and outputs are reversed in Algorithm 1:

$$E^{\lambda}_{\text{data}} \le b\left(mn + \frac{2n(m-(\beta-d))}{d} + n + 2(\beta - d)\right). \tag{25}$$

This upper bound has the smallest value for $d = \beta - 1$, provided that $m \ge \beta$, since $m \ge \beta$ is equivalent to

$$\frac{2n(m-1)}{\beta - 1} \le \frac{2n(m-(\beta-d))}{d}.$$

Finally, assuming $n \ge \beta$ and $m \ge \beta$, we can compare (24) and (25) for their smallest values, namely $d = 1$ and $d = \beta - 1$, respectively:

$$b\left(mn + \frac{m(n-1)}{\beta - 1} + 2m + 1\right) \overset{?}{\le} b\left(mn + \frac{2n(m-1)}{\beta - 1} + n + 2\right) \tag{26}$$

which can be rewritten as

$$0 \overset{?}{\le} m(n - 2\beta + 3) + n(\beta - 3) + \beta - 1\,. \tag{27}$$

This inequality holds for $n > 2\beta - 3$ implying $n \ge \beta$, due to $\beta \ge 2$. Therefore, we can conclude that for sufficiently large $n > 2\beta - 3$ and $m \ge \beta$, the minimal energy for fully-connected layers achieved by the dataflow described in Algorithm 1 is obtained when $d = 1$, i.e., when Buffer is partitioned to $\beta - 1$ outputs, one input, and one weight. This situation leads to the following upper bound:

$$E^{\lambda}_{\text{data}} \le b\left(mn + \frac{m(n-1)}{\beta - 1} + 2m + 1\right). \tag{28}$$

4.2. Bounded number of inputs in Buffer

The second dataflow is parameterized by the maximum number c of inputs that can simultaneously occur in Buffer, where $1 \le c \le \beta - 1$. For notational simplicity, we assume that $\beta - 1$ divides m.

The main idea of this dataflow is that the m outputs are split into $\frac{m}{\beta-1}$ groups. These groups, each of size $\beta-1$ outputs, are read into Buffer one after the other in such a way that the next group overwrites $\beta - c$ outputs of the current group and $c - 1$ out of c inputs currently stored in Buffer. For each such group loaded into Buffer, all the $n - 1$ inputs are read one by one into Buffer so that each of the first $n - c$ of these inputs replaces the previously read input whereas the last $c - 1$ inputs overwrite outputs from the current group. This procedure ensures that all the mn input–output pairs will occur in Buffer containing at most c inputs. This dataflow is illustrated in Fig. 3.

The dataflow is formally described in Algorithm 2 where the comments (beginning with double slashes) specify the current Buffer contents $B_t \subseteq X \cup Y$ after t DRAM read accesses. Thus, the sequence of sets $B_0, B_1, \dots, B_p$ satisfies conditions 1–4 and $|B_t \cap X| \le c$ for every $t = 0, \dots, p$.

At the beginning when Buffer is empty (line 1), the first c inputs are read into Buffer (loop 2–4) so that $B_c = \{x_1, \dots, x_c\}$ (line 4). Then the algorithm continues with the outer *for* loop 6–28 which goes through all the $\frac{m}{\beta-1}$ groups of $\beta - 1$ outputs, indexed as $k = 0, \dots, \frac{m}{\beta-1} - 1$.

Algorithm 2 The dataflow with a bounded number c of inputs in Buffer.

1: // $B_0 = \emptyset$
2: **for** $i = 1$ **to** c **do**
3: **read** x_i into Buffer // $B_i = \{x_1, \dots, x_i\}$
4: **end for** // $B_c = \{x_1, \dots, x_c\}$
5: **for** $k = 0$ **to** $\frac{m}{\beta-1} - 1$ **do**
6: **for** $j = 1$ **to** $\beta - c$ **do**
7: **if** $k = 0$ **then**
8: **read** y_j into Buffer // $B_{c+j} = \{x_1, \dots, x_c, y_1, \dots, y_j\}$
9: **else**
10: **read** $y_{k(\beta-1)+j}$ into Buffer by overwriting $y_{(k-1)(\beta-1)+c+j-1}$
11: // $\{y_{k(\beta-1)+1}, \dots, y_{k(\beta-1)+j}, y_{(k-1)(\beta-1)+c+j}, \dots, y_{k(\beta-1)}\} \subset B_{k(n+\beta-2)+c+j}$
12: **end if**
13: **end for**
// $B_{k(n+\beta-2)+\beta} = \{x_{(k \bmod n)+1}, \dots, x_{((k+c-1) \bmod n)+1}, y_{k(\beta-1)+1}, \dots, y_{k(\beta-1)+\beta-c}\}$
14: **for** $j = \beta - c + 1$ **to** $\beta - 1$ **do**
15: $\ell \leftarrow ((k + \beta - j) \mod n) + 1$
16: **read** $y_{k(\beta-1)+j}$ into Buffer by overwriting x_ℓ
17: **end for** // $B_{k(n+\beta-2)+\beta+c-1} = \{x_{(k \bmod n)+1}, y_{k(\beta-1)+1}, \dots, y_{(k+1)(\beta-1)}\}$
18: **for** $i = n + k$ **downto** $k + c + 1$ **do**
19: $\ell \leftarrow ((i - 1) \mod n) + 1$; $\ell_1 \leftarrow (i \mod n) + 1$
20: **read** x_ℓ into Buffer by overwriting x_{ℓ_1}
21: // $B_{k(n+\beta-2)+n+\beta+k+c-i} = \{x_\ell, y_{k(\beta-1)+1}, \dots, y_{(k+1)(\beta-1)}\}$
22: **end for** // $B_{k(n+\beta-2)+n+\beta-1} = \{x_{((k+c) \bmod n)+1}, y_{k(\beta-1)+1}, \dots, y_{(k+1)(\beta-1)}\}$
23: **for** $i = k + c$ **downto** $k + 2$ **do**
24: $\ell \leftarrow ((i - 1) \mod n) + 1$
25: **read** x_ℓ into Buffer by overwriting $y_{k(\beta-1)+k+c-i+1}$
26: // $B_{k(n+\beta-2)+n+\beta+k+c-i} = \{x_\ell, \dots, x_{((k+c) \bmod n)+1}, y_{k(\beta-1)+c+k-i+2}, \dots, y_{(k+1)(\beta-1)}\}$
27: **end for**
// $B_{(k+1)(n+\beta-2)+c} = \{x_{((k+1) \bmod n)+1}, \dots, x_{((k+c) \bmod n)+1}, y_{k(\beta-1)+c}, \dots, y_{(k+1)(\beta-1)}\}$
28: **end for**

The first $\beta - c$ of these $\beta - 1$ outputs are read into Buffer during the first inner *for* loop 6–13. Namely, for the first group of outputs with the index $k = 0$ (line 7), when Buffer contains only the c inputs $x_1, \dots, x_c$, these $\beta - c$ outputs $y_1, \dots, y_{\beta-c}$ are just read into Buffer (line 8) where there is enough space for them, which means $B_\beta = \{x_1, \dots, x_c, y_1, \dots, y_{\beta-c}\}$ (cf. line 13 for $k = 0$). For the following groups of outputs with the index $k > 0$ (line 9), these $\beta - c$ outputs $y_{k(\beta-1)+1}, \dots, y_{k(\beta-1)+\beta-c}$ are read into Buffer one by one, replacing the $\beta - c$ outputs $y_{(k-1)(\beta-1)+c}, \dots, y_{(k-1)(\beta-1)+\beta-1}$ which remained in Buffer from the previous group with the index $k - 1$ (lines 10–11).

In the following second inner *for* loop 14–17, the remaining $c - 1$ outputs $y_{k(\beta-1)+\beta-c+1}, \dots, y_{k(\beta-1)+\beta-1}$ of the current group with the index $k \ge 0$, are read into Buffer one by one, overwriting the $c - 1$ inputs $x_{((k+c-1) \bmod n)+1}, \dots, x_{(k \bmod n)+2}$ with the decreasing index, respectively (line 16), that are currently stored in Buffer (line 13). This means that only one input $x_{(k \bmod n)+1}$ remains there (line 17). Note that the indices of inputs are shifted by k and looped using the *modulo* function (line 15) so that the nth input is followed by the first one which, on the other hand, is preceded by the nth input. Thus, the whole group of outputs with the index k is then contained in Buffer (line 17 where the index of $B_{k(n+\beta-2)+\beta+c-1}$ for $k > 0$ takes into account also the DRAM accesses

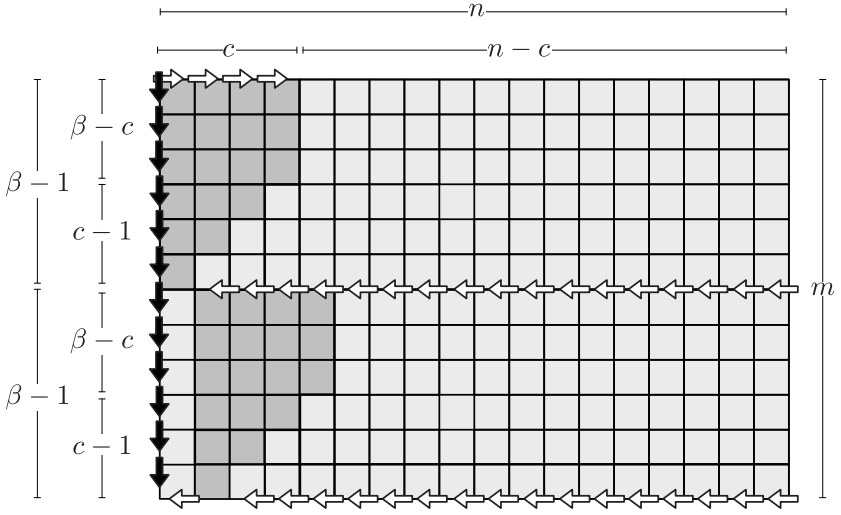

Fig. 3. Illustration of the dataflow with a bounded number c of inputs in Buffer. At the beginning, the first c inputs are read (top horizontal arrows). Afterwards, $\beta - c$ and then $c - 1$ outputs are read (top vertical arrows), which generates the first input–output pairs (top left cells, squared and stair-shaped dark regions, respectively). Note that the readings of the $c - 1$ outputs overwrite $c - 1$ inputs currently stored in Buffer, and hence only generate $\frac{c(c-1)}{2}$ new input–output pairs (stair-shaped dark region). Next, the remaining $n - c$ and the already considered $c - 1$ inputs are read in the reverse order (middle horizontal arrows), all of them yielding novel input–output pairs (top right cells, squared and stair-shaped block, respectively). Note that the last $c - 1$ inputs read, which overwrite $c - 1$ outputs currently stored in Buffer, had already been processed earlier in Buffer and thus generate only $\frac{c(c-1)}{2}$ new input–output pairs (stair-shaped light region). The dataflow continues in this way by reading outputs and inputs alternatively. At each iteration of the outer loop, input readings are shifted by one position in a circular fashion.

in between the readings of the two groups, as described on lines 18–27 and commented below).

The third inner *for* loop 18–22 is used to read $n-c$ inputs into Buffer one by one, starting with $x_{((n+k-1) \bmod n)+1}$ and following the decreasing index, in such a way that each such input replaces the previously read one (lines 19–20). This continues in the last inner *for* loop 23–27 where the remaining $c - 1$ inputs $x_{((k+c-1) \bmod n)+1}, \dots, x_{((k+1) \bmod n)+1}$ with the decreasing index are read into Buffer one by one, overwriting the $c - 1$ outputs $y_{k(\beta-1)+1}, \dots, y_{k(\beta-1)+c-1}$, respectively, from the current group with the index k (lines 24–26).

According to line 13, the first $\beta - c$ outputs $y_{k(\beta-1)+1}, \dots, y_{k(\beta-1)+\beta-c}$ from the kth group meet the $c - 1$ inputs $x_{((k+1) \bmod n)+1}, \dots, x_{((k+c-1) \bmod n)+1}$ in Buffer. The remaining $c - 1$ outputs $y_{k(\beta-1)+\beta-c+1}, \dots, y_{(k+1)(\beta-1)}$ from this group occur in Buffer simultaneously with these $c - 1$ inputs, as stated in line 27. The whole kth group of outputs $y_{k(\beta-1)+1}, \dots, y_{(k+1)(\beta-1)}$ meets the input $x_{(k \bmod n)+1}$ in Buffer after the loop 14–17 is performed (line 17), while each of the remaining $n - c$ inputs occurs at the same time with this group in Buffer during the loop 18–22 (line 21). This is repeated for every group of outputs (outer loop 5–28), which guarantees that all the mn input–output pairs will occur in Buffer.

We will calculate the number p of DRAM read accesses in the dataflow described by Algorithm 2. After the first c inputs are read into Buffer in the loop 2–4, the outer loop 5–28 which runs $\frac{m}{\beta-1}$ times, includes $\beta - c$ and $c - 1$ DRAM accesses to read outputs in the inner loops 6–13 and 14–17, respectively, and $n - c$ and $c - 1$ DRAM accesses for reading inputs in the inner loops 18–22 and 23–27, respectively. Altogether, we have

$$p = c + \frac{m}{\beta - 1}\left((\beta - c) + (c - 1) + (n - c) + (c - 1)\right) = \frac{m(n-1)}{\beta - 1} + m + c\,. \quad (29)$$

Hence, this dataflow provides an upper bound on the data energy of a fully-connected layer λ:

$$E^{\lambda}_{\text{data}} \leq b\left(mn + \frac{m(n-1)}{\beta - 1} + 2m + c\right) \quad (30)$$

according to (5), (21), and (22). Note that Algorithm 1 for $d = 1$ coincides with Algorithm 2 for $c = 1$, producing the same upper bound (28).

This upper bound (28) can be compared to the general lower bound (20) on the data energy which is still smaller by the linear additive term $\frac{1}{2}m$. The lower bound will be improved in some special cases in Section 5. Nevertheless, we have achieved the asymptotically optimal quadratic data energy complexity of evaluating a fully-connected layer in terms of the number of its inputs and outputs.

5. Optimal energy complexity for a partitioned Buffer

We now study the case where Buffer is divided into two separated parts dedicated to the reading of d inputs and $\beta - d$ outputs, respectively, plus one float for weights, where d is a fixed parameter such that $1 \leq d \leq \beta - 1$. In this context, we improve the general lower bound (20) on the data energy $E^{\lambda}_{\text{data}}$ of a fully-connected layer λ so that it matches the upper bounds (24) and (25), up to an additive constant. We investigate two cases according to whether d is at most or at least $\frac{2}{3}\beta$.

***Case* $1 \leq d \leq \frac{2}{3}\beta$.** First assume that

$$1 \leq d \leq \tfrac{2}{3}\beta\,. \quad (31)$$

We formulate a linear program for finding μ and ν that

$$\text{minimize} \quad 2\mu + \nu \quad (32)$$

$$\text{subject to} \quad d\mu + (\beta - d)\nu \geq mn \quad (33)$$

$$\mu \geq m \quad (34)$$

$$\nu \geq 0\,, \quad \mu \geq 0\,. \quad (35)$$

The constraint (33) follows from the requirement that all the mn input–output couples have to occur in Buffer, since by reading one output or input, at most d or $\beta - d$ new pairs meet in Buffer, respectively. The constraint (34) follows from the fact that at least m outputs are read into Buffer. We convert the linear program (32)–(35) to the corresponding dual linear program for finding ϕ and ψ that

$$\text{maximize} \quad mn\,\phi + m\,\psi \quad (36)$$

$$\text{subject to} \quad d\phi + \psi \leq 2 \quad (37)$$

$$(\beta - d)\phi \leq 1 \quad (38)$$

$$\phi \geq 0, \quad \psi \geq 0\,. \quad (39)$$

Observe that $\phi_0 = \frac{1}{\beta - d}$ and $\psi_0 = 2 - \frac{d}{\beta - d}$ is a feasible solution for the dual program, satisfying (37)–(39) due to (31).

By the weak duality theorem, the objective function value of the primal (32) at any feasible solution is lower bounded by the objective function value of the dual (36) at any feasible solution, that is,

$$2\mu + \nu \geq mn\,\phi_0 + m\,\psi_0 = \frac{m(n-d)}{\beta - d} + 2m\,. \tag{40}$$

According to (5), the inequality (40) provides the following lower bound on the data complexity of a fully-connected layer λ:

$$E^{\lambda}_{\text{data}} \geq b\left(mn + \frac{m(n-d)}{\beta - d} + 2m\right) \tag{41}$$

when Buffer is divided into two parts for d inputs and $\beta - d$ outputs, and the fixed parameter d meets (31). This lower bound matches the corresponding upper bound (24) achieved by the dataflow described in Algorithm 1, up to the additive constant d.

Case $\frac{2}{3}\beta \leq d \leq \beta - 1$. Similarly, for

$$\tfrac{2}{3}\beta \leq d \leq \beta - 1\,, \tag{42}$$

we have a linear program for finding μ and ν that minimize $2\mu + \nu$ subject to $d\mu + (\beta - d)\nu \geq mn$, $\nu \geq n$, $\nu \geq 0$, and $\mu \geq 0$. This is converted to the corresponding dual linear program for finding ϕ and ψ that maximize $mn\,\phi + n\,\psi$ subject to $d\phi \leq 2$, $(\beta - d)\phi + \psi \leq 1$, $\psi \geq 0$, and $\psi \geq 0$, which has a feasible solution $\phi_1 = \frac{2}{d}$ and $\psi_1 = 1 - \frac{2(\beta-d)}{d}$ due to (42).

By the weak duality theorem we have

$$2\mu + \nu \geq mn\,\phi_1 + n\,\psi_1 = \frac{2n(m - (\beta - d))}{d} + n \tag{43}$$

which provides the following lower bound on the data complexity of a fully-connected layer λ:

$$E^{\lambda}_{\text{data}} \geq b\left(mn + \frac{2n(m - (\beta - d))}{d} + n\right) \tag{44}$$

when Buffer is divided into two parts for d inputs and $\beta - d$ outputs, and the fixed parameter d meets (42). This lower bound matches the corresponding upper bound (25) achieved by the dataflow described in Algorithm 1 with the reversed role of inputs and outputs, up to the additive constant $2(\beta - d)$.

We can conclude that the data energy for fully-connected layers achieved by the dataflow described in Algorithm 1 when Buffer is partitioned into d inputs, $\beta - d$ outputs, and one weight, is optimal for any fixed d, and the minimum of data energy (28) is achieved for $d = 1$.

6. Experimental validation

In this section, we compare the theoretical energy complexity introduced in Section 2 to the real energy consumption estimated by the Timeloop/Accelergy software tool for evaluating DNN accelerator designs. The Timeloop (Parashar et al., 2019) finds a mapping of a convolutional layer specified by its parameters (e.g. height, width, depth, kernel size, stride) onto a given hardware platform, which is optimal in terms of energy consumption estimated by Accelergy (Wu et al., 2019) reporting the energy statistics. Here, we employ the tool for fully-connected layers as a special case of convolutional layers where the feature maps are reduced to single neurons.

In particular, the Timeloop can design the hardware architecture parameters, the approach of how a layer is mapped to hardware, how memory caches are used, and so on. In energy optimization, Timeloop performs design space exploration to find a (sub)optimal configuration for a given layer. Since there are a reasonable number (tens to hundreds of thousands) of configurations for some layers and architectures (such as Eyeriss), Timeloop performs a brute-force search with a guaranteed optimum. Otherwise, an internal heuristic is involved to limit unsuitable paths (e.g. Simba accelerator), which can be replaced, for example, by a genetic algorithm. For energy evaluation, the Timeloop calls the Accelergy tool which determines the exact number of clock cycles required to process inference, the number of memory accesses for reads and writes, and other metrics. From these, the overall power consumption of the layer inference is determined based on parameters obtained from hardware synthesis for application-specific integrated circuits (such as Synopsys Design Compiler[1] or CACTI[2] for memories).

We have employed Simba (Shao et al., 2019) and Eyeriss (Chen et al., 2016) as the target hardware platforms onto which fully-connected layers with increasing number of inputs, outputs, and weights have been mapped. These two hardware architectures have been chosen as prominent examples of modern DNN inference accelerators which are general and not tied to a specific DNN as is common in single-purpose accelerators with FPGAs (Mittal, 2020) or printed electronics. They are based on a systolic array of processing elements which communicate with each other without the need for additional memory accesses. This represents a state-of-the-art approach widely used in many real-world DNN inference accelerators such as ARM Cortex-M Processor (Orășan et al., 2022), TPU, etc. Nevertheless, other approaches can be used to support our energy complexity model. All configuration files used in experiments are publicly available at Github.[3]

For a fully-connected layer λ, we measure empirical dependencies of the optimal *data energy* independently on its number of inputs n, outputs m, and weights mn, which is minimized by using the Timeloop/Accelergy tool for the Simba and Eyeriss architectures. These dependencies are then compared to the corresponding upper bound (28) on $E^{\lambda}_{\text{data}}$ achieved by the dataflows in the energy complexity model as presented in Section 4, which matches asymptotically the quadratic lower bound (20) in terms of n and m, as was proven in Section 3. In particular, for the comparison of empirical energy consumptions to the theoretical data energy $E^{\lambda}_{\text{data}}$, we use the following asymptotic optimal bounds:

$$E^{\lambda}_{\text{data}} = \Theta(n)\,, \quad E^{\lambda}_{\text{data}} = \Theta(m)\,, \quad E^{\lambda}_{\text{data}} = \Theta(mn)\,, \tag{45}$$

which are derived from (20) and (28) for individual variables (when the other independent parameter is considered to be constant).

Fig. 4 presents the results of experimental comparison of energy-efficient CNN hardware implementations to our theoretical energy complexity model separately for individual parameters of fully-connected layers. By using the Timeloop/Accelergy tool applied to the Simba and Eyeriss hardware architectures, the optimal values of their data energy consumption have been estimated for a fully-connected layer λ with increasing parameters n, m, and mn, each separately. In order to make the experiment computationally feasible, 32 values for n (the same for m) were taken from the interval 128 to 4096 with the step 128, whereas the other parameter m (respectively n) was fixed at the value of 1024, which represents realistic sizes of fully-connected DNN layers. For the number of weights mn, we took all possible 32×32 pairs of these values for m and n.

These parameters serve as independent variables in regression analysis where the relationships between the data energy and the independent variables are modeled as functions with asymptotics (45), including multiplicative and additive coefficients c_2 and c_1, respectively. As depicted in Fig. 4, these coefficients are approximated by the method of least squares so that the theoretical data energy $E^{\lambda}_{\text{data}}$ (dashed lines) fits energy estimates by Timeloop/Accelergy (displayed by bars), which confirms the asymptotic trends (45) in the energy complexity model.

In addition, the energy complexity model has been validated by statistical tests using quadratic regression with the function model

[1] https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[2] https://github.com/HewlettPackard/cacti

[3] https://github.com/PetraVidnerova/timeloop-accelergy-test

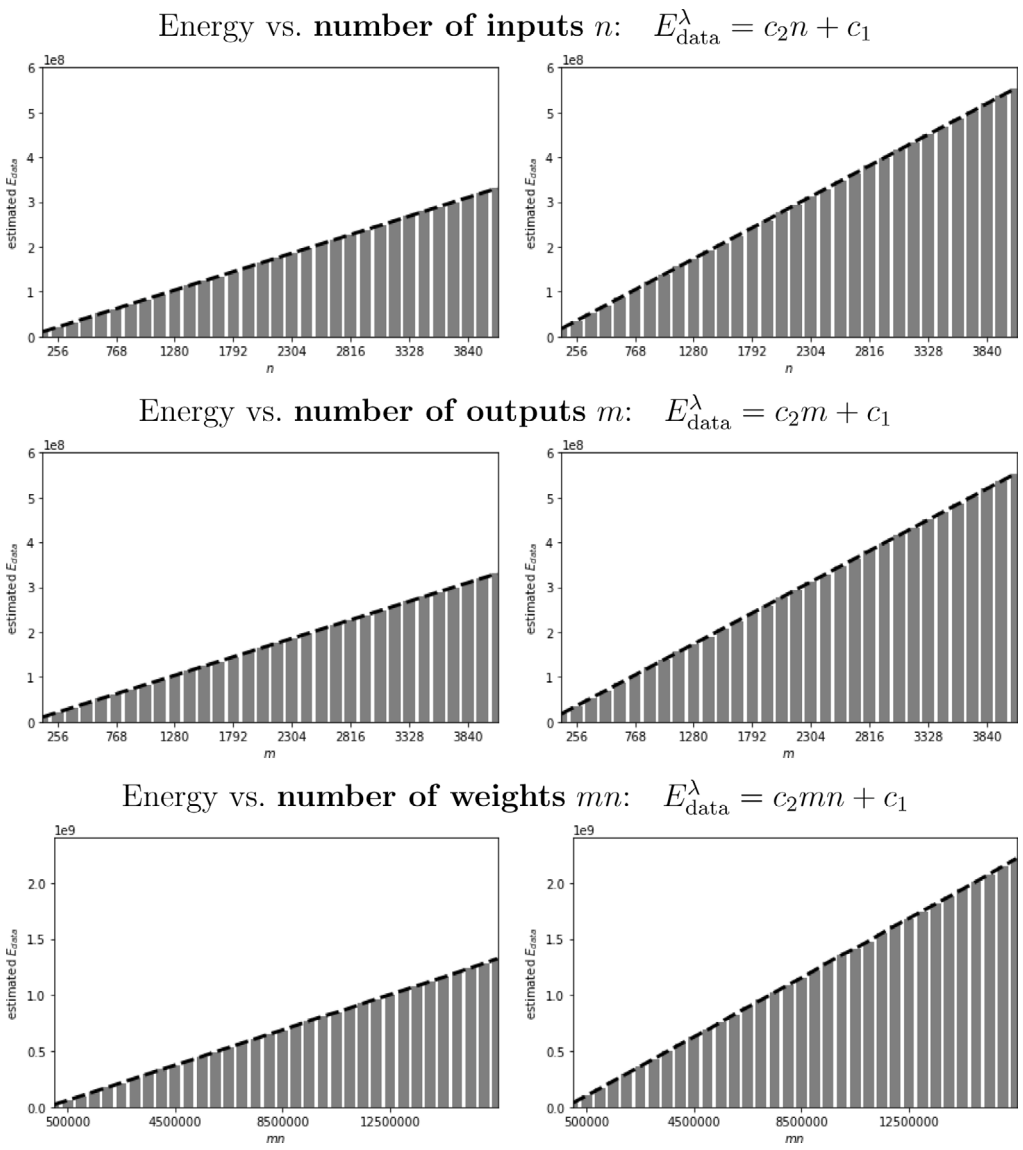

Fig. 4. The data energy estimates by Timeloop/Accelergy (displayed by bars) for a fully-connected layer λ with increasing parameters n, m, and mn, each separately (from top to bottom), on the Simba (left) and Eyeriss (right) architectures, which fit the asymptotic trends (45) in the energy complexity model (dashed lines).

$ax^2 + bx + c$ for the independent variable x to be n, m, and mn, respectively. These statistical tests have approved the linearity in n, m, and mn, with the p-values 0.2447, 0.6468, and 0.0575, respectively, for Simba, and 0.1494, 0.4801, and 0.0531, respectively, for Eyeriss, accepting the null hypothesis of $a = 0$ (at the significance level 0.05) in all these cases.

The presented experiments have thus validated the energy complexity model whose upper and lower bounds on theoretical energy for fully-connected layers fit asymptotically very well the energy consumption estimated by the Timeloop/Accelergy tool for the Simba and Eyeriss hardware platforms.

7. Conclusion

In this paper, we have theoretically analyzed the energy complexity model for CNNs introduced in our previous work (Šíma et al., 2024) which was shown to be asymptotically consistent with the energy consumption estimates of their various hardware implementations. We have restricted ourselves to fully-connected layers, which constitute the most common blocks of DNNs, and plan to extend this analysis to the case of convolutional layers.

We have shown a general lower bound on energy complexity of fully-connected layers. We have presented two dataflows for fixed and bounded numbers of inputs residing in Buffer, respectively, and calculated their energy costs to obtain upper bounds on energy complexity. We have proven the matching lower bound on the energy for the first dataflow, which in turn, provides the optimal energy complexity for fully-connected layers in the case where Buffer is partitioned into two separate parts for inputs and outputs.

Since the presented general lower and upper bounds differ only in a linear additive term, we have thus achieved the asymptotically optimal quadratic energy complexity of evaluating a fully-connected layer in terms of the number of its inputs and outputs. This asymptotic quadratic energy complexity has been experimentally confirmed by the real energy consumption estimates for the Simba and Eyeriss hardware architectures, using the Timeloop/Accelergy software tool.

We conjecture that the general lower bound on energy complexity of fully-connected layers can be improved to match the presented upper bound, which constitutes a path for future work. The main challenge is to generalize this analysis to the case of convolutional layers in order to achieve their optimal energy complexity.

CRediT authorship contribution statement

Jiří Šíma: Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Jérémie Cabessa:** Writing – review & editing, Visualization, Validation, Formal analysis. **Petra Vidnerová:** Writing – review & editing, Visualization, Validation, Software, Data curation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

The presentation of this paper benefited from valuable suggestions of anonymous reviewers. This work was supported by the Czech Science Foundation grant GA22-02067S and the institutional support RVO: 67985807. We thank Petr Savický for inspiring discussions in the early stages of this research and Jan Kalina for expert consultation regarding statistical tests.

References

Alwani, M., Chen, H., Ferdman, M., & Milder, P. A. (2016). Fused-layer CNN accelerators. In *Proceedings of the 49th annual IEEE/ACM international symposium on microarchitecture*. MICRO 2016, Article 22. http://dx.doi.org/10.1109/MICRO.2016.7783725.

Ansari, M. S., Mrazek, V., Cockburn, B. F., Sekanina, L., Vasicek, Z., & Han, J. (2020). Improving the accuracy and hardware efficiency of neural networks using approximate multipliers. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *28*(2), 317–328. http://dx.doi.org/10.1109/TVLSI.2019.2940943.

Armeniakos, G., Zervakis, G., Soudris, D., & Henkel, J. (2023). Hardware approximate techniques for deep neural network accelerators: A survey. *ACM Computing Surveys*, *55*(4), Article 83. http://dx.doi.org/10.1145/3527156.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, & H. Lin (Eds.), *Advances in neural information processing systems: Proceedings of the 34th anual conference on neural information processing systems*: *vol. 33*, NeurIPS 2020, (pp. 1877–1901). URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Chen, Y., Emer, J. S., & Sze, V. (2016). Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd annual ACM/IEEE international symposium on computer architecture* ISCA 2016, (pp. 367–379). http://dx.doi.org/10.1109/ISCA.2016.40.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., ... Fiedel, N. (2023). PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, *24*(240), 1–113, URL http://jmlr.org/papers/v24/22-1144.html.

Deng, L., Li, G., Han, S., Shi, L., & Xie, Y. (2020). Model compression and hardware acceleration for neural networks: A comprehensive survey. *Proceedings of the IEEE*, *108*(4), 485–532. http://dx.doi.org/10.1109/JPROC.2020.2976475.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., & Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the 9th international conference on learning representations*. ICLR 2021, URL https://openreview.net/forum?id=YicbFdNTTy.

Gonthier, M., Marchal, L., & Thibault, S. (2023). Taming data locality for task scheduling under memory constraint in runtime systems. *Future Generation Computer Systems*, *143*, 305–321. http://dx.doi.org/10.1016/J.FUTURE.2023.01.024.

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In F. Bach, & D. Blei (Eds.), *JMLR workshop and conference proceedings*: *vol. 37*, *Proceedings of the 32nd international conference on machine learning* ICML 2015, (pp. 1737–1746). URL http://proceedings.mlr.press/v37/gupta15.html.

Huang, X., Khetan, A., Cvitkovic, M., & Karnin, Z. S. (2020). TabTransformer: Tabular data modeling using contextual embeddings. http://dx.doi.org/10.48550/arXiv.2012.06678, CoRR, arXiv:2012.06678 [cs.LG].

Jouppi, N. P., Young, C., Patil, N., & Patterson, D. A. (2018). A domain-specific architecture for deep neural networks. *Communications of the ACM*, *61*(9), 50–59. http://dx.doi.org/10.1145/3154484.

Li, Z., Li, H., & Meng, L. (2023). Model compression for deep neural networks: A survey. *Computers*, *12*(3), Article 60. http://dx.doi.org/10.3390/COMPUTERS12030060.

Luccioni, A. S., Viguier, S., & Ligozat, A.-L. (2022). Estimating the carbon footprint of BLOOM, a 176b parameter language model. http://dx.doi.org/10.48550/arXiv.2211.02001, CoRR, arXiv:2211.02001 [cs.LG].

Lyu, Z., Yu, T., Pan, F., Zhang, Y., Luo, J., Zhang, D., Chen, Y., Zhang, B., & Li, G. (2023). A survey of model compression strategies for object detection. *Multimedia Tools and Applications*, *83*, 48165–48236. http://dx.doi.org/10.1007/s11042-023-17192-x.

Mishra, R., Gupta, H. P., & Dutta, T. (2020). A survey on deep neural network compression: Challenges, overview, and solutions. http://dx.doi.org/10.48550/arXiv.2010.03954, CoRR, arXiv:2010.03954 [cs.LG].

Mittal, S. (2016). A survey of techniques for approximate computing. *ACM Computing Surveys*, *48*(4), Article 62. http://dx.doi.org/10.1145/2893356.

Mittal, S. (2020). A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computing and Applications*, *32*(4), 1109–1139. http://dx.doi.org/10.1007/s00521-018-3761-1.

Mittal, S., Verma, G., Kaushik, B., & Khanday, F. A. (2021). A survey of SRAM-based in-memory computing techniques and applications. *Journal of Systems Architecture*, *119*, Article 102276. http://dx.doi.org/10.1016/J.SYSARC.2021.102276.

Orășan, I. L., Seiculescu, C., & Căleanu, C. D. (2022). A brief review of deep neural network implementations for ARM Cortex-M processor. *Electronics*, *11*(16), Article 2545. http://dx.doi.org/10.3390/electronics11162545.

Parashar, A., Raina, P., Shao, Y. S., Chen, Y., Ying, V. A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S. W., & Emer, J. S. (2019). Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the IEEE international symposium on performance analysis of systems and software* ISPASS 2019, (pp. 304–315). http://dx.doi.org/10.1109/ISPASS.2019.00042.

Shao, Y. S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N. R., Raina, P., Tell, S. G., Zhang, Y., Dally, W. J., Emer, J. S., Gray, C. T., Khailany, B., & Keckler, S. W. (2019). Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture* MICRO 2019, (pp. 14–27). http://dx.doi.org/10.1145/3352460.3358302.

Silvano, C., Ielmini, D., Ferrandi, F., Fiorin, L., Curzel, S., Benini, L., Conti, F., Garofalo, A., Zambelli, C., Calore, E., Schifano, S. F., Palesi, M., Ascia, G., Patti, D., Perri, S., Petra, N., Caro, D. D., Lavagno, L., Urso, T., ... Birke, R. (2023). A survey on deep learning hardware accelerators for heterogeneous HPC platforms. CoRR, arXiv:2306.15552 [cs.AR].

Šíma, J., & Cabessa, J. (2023). Energy complexity of fully-connected layers. In I. Rojas, G. Joya, & A. Catala (Eds.), *LNCS*: *vol. 14134, part I*, *Proceedings of the 17th international work-conference on artificial neural networks* IWANN 2023, (pp. 3–15). Springer, http://dx.doi.org/10.1007/978-3-031-43085-5_1.

Šíma, J., Vidnerová, P., & Mrázek, V. (2024). Energy complexity of convolutional neural networks. *Neural Computation*, http://dx.doi.org/10.1162/neco_a_01676.

Sze, V., Chen, Y., Yang, T., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, *105*(12), 2295–2329. http://dx.doi.org/10.1109/JPROC.2017.2761740.

Sze, V., Chen, Y., Yang, T., & Emer, J. S. (2020). Efficient processing of deep neural networks. *Synthesis lectures on computer architecture*, Morgan & Claypool Publishers, http://dx.doi.org/10.2200/S01004ED1V01Y202004CAC050.

Tang, Y., Wang, Y., Guo, J., Tu, Z., Han, K., Hu, H., & Tao, D. (2024). A survey on transformer compression. http://dx.doi.org/10.48550/arXiv.2402.05964, CoRR, arXiv:2402.05964 [cs.LG].

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA: open and efficient foundation language models. http://dx.doi.org/10.48550/arXiv.2302.13971, CoRR, arXiv:2302.13971 [cs.CL].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, & R. Garnett (Eds.), *Advances in neural information processing systems: Proceedings of the 31st annual conference on neural information processing systems*: *vol. 30*, NIPS 2017, (pp. 5998–6008). URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Wu, Y. N., Emer, J. S., & Sze, V. (2019). Accelergy: An architecture-level energy estimation methodology for accelerator designs. In D. Z. Pan (Ed.), *Proceedings of the IEEE/ACM international conference on computer aided design*. ICCAD 2019, http://dx.doi.org/10.1109/ICCAD45719.2019.8942149.

Yang, T., Chen, Y., Emer, J. S., & Sze, V. (2017). A method to estimate the energy consumption of deep neural networks. In M. B. Matthews (Ed.), *Proceedings of the IEEE 51st asilomar conference on signals, systems, and computers* ACSSC 2017, (pp. 1916–1920). http://dx.doi.org/10.1109/ACSSC.2017.8335698.

Zhou, G., Zhou, J., & Lin, H. (2018). Research on NVIDIA deep learning accelerator. In *Proceedings of the 12th IEEE international conference on anti-counterfeiting, security, and identification* ASID 2018, (pp. 192–195). http://dx.doi.org/10.1109/ICASID.2018.8693202.