Author(s): Klemetti, Antti; Raatikainen, Mikko; Myllyaho, Lalli; Mikkonen, Tommi; Nurminen, Jukka K.

Title: Systematic Literature Review on Cost-efficient Deep Learning

Year: 2023

Version: Published version

Please cite the original version:

Klemetti, A., Raatikainen, M., Myllyaho, L., Mikkonen, T., & Nurminen, J. K. (2023). Systematic Literature Review on Cost-efficient Deep Learning. IEEE Access, 11, 90158-90180. https://doi.org/10.1109/access.2023.3275431

IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received 3 March 2023, accepted 2 May 2023, date of publication 11 May 2023, date of current version 28 August 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3275431

TOPICAL REVIEW

Systematic Literature Review on Cost-Efficient Deep Learning

ANTTI KLEMETTI[1], MIKKO RAATIKAINEN[1], LALLI MYLLYAHO[1], TOMMI MIKKONEN[2], AND JUKKA K. NURMINEN[1], (Member, IEEE)

[1]Department of Computer Science, Faculty of Science, University of Helsinki, 00014 Helsinki, Finland
[2]Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland

Corresponding author: Antti Klemetti (antti.klemetti@helsinki.fi)

This work was supported by the Industrial Machine Learning for Enterprises (IML4E) Project of ITEA4 funded by Business Finland.

ABSTRACT Cloud computing and deep learning, the recent trends in the software industry, have enabled small companies to scale their business up rapidly. However, this growth is not without a cost – deep learning models are related to the heaviest workloads in cloud data centers. When the business grows, the monetary cost of deep learning in the cloud also grows fast. Deep learning practitioners should be prepared and equipped to limit the growing cost. We emphasize monetary cost instead of computational cost although often the same methods decrease both types of cost. We performed a systematic literature review on the methods to control the cost of deep learning. Our library search resulted in 16,066 papers from three article databases, IEEE Xplore, ACM Digital Library, and Scopus. We narrowed them down to 112 papers that we categorized and summarized. We found that: 1) Optimizing inference has raised more interest than optimizing training. Widely used deep learning libraries already support inference optimization methods, such as quantization, pruning, and teacher-student. 2) The research has been centered around image inputs, and there seems to be a research gap for other types of inputs. 3) The research has been hardware-oriented, and the most typical approach to control the cost of deep learning is based on algorithm-hardware co-design. 4) Offloading some of the processing to client devices is gaining interest and can potentially reduce the monetary cost of deep learning.

INDEX TERMS Cloud computing, cost-efficiency, cost reduction, deep learning, deep neural network, edge offloading, machine learning, systematic literature review.

I. INTRODUCTION

Machine learning (ML) has matured into accurate and popular technology that is widely applied in the software industry. Deep Learning (DL) is a popular approach to ML that has seen significant advances during the past decade. At the same time, cloud computing, often offered by technology giants such as Amazon, Google, and Microsoft [1], has become a commonly applied architectural and deployment paradigm for digital systems. Cloud vendors offer computing resources from their data centers to other companies. Client software typically runs on client-side platforms, in general-purpose web browsers, or selected mobile platforms, such as Android and iOS in the case of mobile phones, which enable a broad reach of users with a manageable number of client application versions. With the advent of the cloud, edge computing has also emerged to offload computations from the cloud closer to the client devices [2]. Broadly, edge refers to anything deployed outside the cloud that is typically interpreted as client devices, such as mobile phones, micro-controllers, and any IoT devices. Alternatively, the edge is seen as a separate processing layer between the client devices and the cloud, also known as fog [3], such as a particular edge data center or a telecommunication operator-hosted setup in mobile network base stations as MEC (Mobile Edge Computing or Multi-access Edge Computing).

The combination of client-side platforms, cloud computing, and DL has introduced new business opportunities enabling companies with only a handful of software developers to grow their businesses from zero to a global scale.

The associate editor coordinating the review of this manuscript and approving it for publication was Michael Lyu.

Creating native optimizations per device type is often not feasible in practice, because the hardware in client devices is heterogeneous [4]. Instead, cloud and web-based server or mobile platform-based client architectures fit most use cases. Moreover, they scale well enough to empower growing businesses without needing to develop or own hardware or hire hardware specialists.

As a downside, industrial DL models require at least one order of magnitude more cloud resources than linear ML models [5]. DL models can have millions of trainable parameters [6], and model execution involves multidimensional matrix operations that are intensive in terms of memory access and computations. With advances in DL and the complexity of problems, the matrix sizes of DL models have grown beyond the on-chip memory of general-purpose processors – even those provisioned from the cloud. Off-chip memory access consumes orders of magnitude more energy than on-chip memory access [7] and dominates the execution time when hitting the memory wall of the currently applied, Von Neumann computer architecture [8]. In fact, the heaviest computing in cloud datacenters is related to DL [9]. The growing size of DL models leads to hitting the memory wall, which leads to slower model execution. This in turn leads to the need to provision more virtual machines (VM) in the cloud, which increases the cost.

The meaning of costs differs among stakeholders. For a computer scientist, the cost usually refers to computational cost, that is, the amount of computing resources and duration of using those resources. For people working in finance, costs refer to the operational or monetary cost, that is, the money spent. For example, offloading computations from the cloud to client devices may increase the computational cost owing to the increased network traffic, but the monetary cost decreases when processing is performed by the end-user devices instead of the cloud. In the scope of our work, cost means the monetary cost, not the computational cost, unless otherwise mentioned.

Using DL to fuel business growth makes sense only if the profit gained is greater than the cost of DL. The cost of DL in the cloud can increase rapidly as business grows for the following reasons:

- Inference frequency increases. Prediction systems for large online companies, such as Facebook, handle tens of trillions of inference requests per day [5], and inference must be performed in near real-time.
- More data is available for training. It makes sense to use this data as there is empirical evidence that DL model accuracy improves as a power-law function of the training set size [10]. A longer retention of historical data and longer training cycles lead to increased cloud costs.
- Models must be frequently retrained to maintain their accuracy under concept drift [11].

DL practitioners should be prepared and equipped for the developments listed above. Although cloud computing can scale, scaling implies additional costs. For example, increasingly frequent training combined with a growing training set size can be addressed by distributing the training to a cluster of machines [12]. However, distributed training is less cost-efficient than training using a single machine. Distributed training involves parameter synchronization overhead and can lead to under-utilized cloud VMs waiting for parameter updates [13]. Likewise, the combination of increased inference frequency and near real-time requirements can be resolved by provisioning more cloud VMs. However, inference requests are not always evenly distributed over time but can come in bursts, resulting in idle resources, communication overheads, and continuous scaling. Therefore, scaling computing is only one of the solutions. In this paper, we review methods that DL practitioners can use to control and limit the monetary cost of DL. For DL researchers, we identify research gaps in cost-efficient DL.

Our research problem addresses the monetary cost-efficiency of a software development organization that adheres to the state of the practice development approach for DL, that is, a cloud is used if a server is needed and clients are based on either web technology or applications for up to a few major platforms. Thus, we excluded dedicated hardware-based cost-efficiency solution proposals. We conducted a systematic literature review [14] that analyzed 112 scientific papers to understand the methods that exist to reduce and control the costs of DL. Our main findings are:

- Optimizing inference has attracted more interest than optimizing training. Widely used deep learning libraries already support some inference optimization methods, such as quantization, pruning, and teacher-student.
- Research has been heavily centered on image inputs leaving a gap for other types of inputs.
- The research has been hardware-oriented, and the most typical approach to control the cost of DL is a proposal based on algorithm-hardware co-design.
- The offloading of DL processing to client devices is gaining interest and can reduce the monetary cost of deep learning.

The remainder of this paper is organized as follows. Section II introduces the structure of the commonly used DL models and walks through surveys related to our work. Section III describes our research method and research questions. Section IV gives an overview of the results. Section V contains a detailed description of all the methods used to reduce and control the cost of DL found via a literature search. Section VI answers the research questions and discusses the findings. Section VII presents the potential threats to the validity of our research. Finally, section VIII presents the conclusions.

II. BACKGROUND AND RELATED WORK

The background of this work consists of introducing Deep Neural Networks (DNN), which are the type of models used in DL. Additionally, we provide insights into related studies.

A. DEEP NEURAL NETWORKS

DNNs have existed for decades, but the most significant leaps have occurred over the past ten years. Deep learning research was sparked in 2012 by a DNN named AlexNet, which won the ImageNet challenge with a two-digit margin [6]. The most popular neural network types that differ by their network structure are Multi-Layer Perceptron (MLP), Convolutional Neural Network (CNN), and Recurrent Neural Network (RNN) [9].

MLPs are general-purpose DNNs with only fully connected layers. A neuron performs a multiply-accumulate (MAC) operation with its inputs using the trained weights for the neuron that outputs a single numeric value. The connection between two neurons describes how the output of a neuron, the numeric value, acts as one of the inputs for a neuron in the next layer. The first layer in a DNN is called the input layer, and the last layer is called the output layer. The layers between them are called the hidden layers. Hidden layers make DL models deeper than linear models with only input and output layers.

A simple MLP with one input layer, one hidden layer, and one output layer is shown in Fig.1. Realistic MLPs have numerous hidden layers and wider input layers. A neuron from the hidden layer is refined on the right side of the figure. Neurons in the hidden and output layers are followed by an activation function that performs a non-linear mapping on the result of the MAC operation from the neuron. Commonly used activation functions include, Sigmoid for binary classifier output, Softmax for multi-class classifier output, and Rectified Linear Unit (ReLU) for the hidden layers. For simplicity, the bias term added to the result of the MAC operation has been omitted.

CNNs have been invented for image processing [15]. For example, AlexNet is a CNN with 60 million 32-bit floating point parameters. CNNs consist of three types of layers: convolutional, pooling, and fully connected layers. Convolutional layers have filters that are – typically small – matrices used to extract patterns, such as edges in images, from the input. The filters are stacked as channels. When the inputs are images, the first convolution layer has three channels – red, green, and blue – for the respective color channels in the image. The subsequent convolutional layers can have any number of channels. Pooling layers are used to reduce the dimensions of the CNNs. A pooling layer splits the input matrix into tiles, and the values in each tile are reduced to one value, for example, the average value within the tile or the maximum value within the tile. Owing to their well-defined structure, convolutional and pooling layers are obvious targets for on-chip memory reuse and computation optimizations.

RNNs are MLPs that have cyclic connections between layers and can maintain a state between subsequent inputs. Using the saved state renders RNNs suitable for machine translation, speech recognition, and time series data processing.

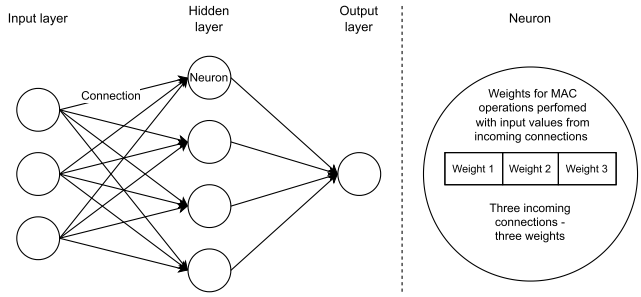

FIGURE 1. A simple MLP and a neuron from the hidden layer.

DNNs can be used as regressors and classifiers. The regressors output a single numeric prediction, and the classifiers output the predicted class. The predicted class has the highest probability, that is, the highest confidence score. The confidence scores are used by several cost-efficient DL methods in this study.

Generative Adversarial Networks (GAN) are DNNs with inverted input and output sizes [16]. For a GAN that generates images, the input could be the classification value, such as the string "cat", and the output would be a picture of the requested entity, such as a cat figure. For a GAN that generates text, the input can be a short question and the output can be a more verbose answer. Training GANs can be computationally heavier than training other types of DNNs as two competing models are trained: a generator and a discriminator. When inferencing in GANs, only the generator is used, but the data flow is still different from that of other types of DNNs, typically from a smaller input to a larger output.

An emerging DNN type is the Graph Neural Network (GNN), which typically takes subgraphs and their properties as input and predicts further properties of the subgraph as output. GNNs can be computationally heavier than MLPs owing to the graph dimensions of the input. Graph Convolutional Networks (GCN) are an implementation of GNNs [17]. In GCNs, graph connections are modeled with an adjacency matrix. GCNs multiply the activations and weights with the adjacency matrix in forward propagation. When the modeled graph becomes large, the adjacency matrix becomes large and typically sparse leading to memory bandwidth and computational challenges similar to those of large weight and activation matrices.

B. RELATED STUDIES

The focus of DL research has shifted from optimizing the prediction accuracy toward green artificial intelligence [18] and understanding the energy consumption of ML models [19]. Some surveys partially overlap with our work, but differ in their research methods and focus. First, this study is a systematic review, unlike the overlapping surveys. However, none of the surveys focused on reducing the monetary cost of DL using software. The hardware acceleration offered in 2018 in public clouds is reviewed in [20]. The Field-Programmable

Gate Array (FPGA) design landscape and usage for DL are summarized in [21]. Most other surveys are about hardware and software co-design [22], [23], [24], [25], [26], [27], [28]. Concerning specific techniques, pruning and quantization are reviewed extensively in [29], and quantization by itself in [30]. The combination of ML and edge computing, called edge intelligence or edge AI, is discussed in [24], [31], and [32].

III. RESEARCH METHOD

The research method of this study follows the Systematic Literature Review approach [14].

A. RESEARCH PROBLEM AND QUESTIONS

Our research problem addresses the cost-efficiency of an organization that adheres to the state of the practice development approach for DL, that is, a cloud is used if a server is needed and clients are based on either web technology or applications for up to a few major platforms. Alternatively, an organization develops embedded systems, but even then, general-purpose hardware is often used, and software development is not significantly different. Consequently, novel DL-specific hardware-based solutions for cost-efficiency are mainly out of our scope because we cannot expect the solutions to be applicable in practice. That is, DL-specific hardware development, manufacturing, and support require different expertise and resources than software development and introduce different costs that may not be desirable. However, we included those hardware-based solutions that are currently available or are promising based on the popularity that might soon be realized in commercial offerings. For example, tensor processing units (TPU) as DL-specific hardware became surprisingly rapidly available for business. Consequently, our research problem is as follows:

> *How can a software development organization reduce its DL costs?*

We refine the research problem to the following more precise research questions:

- RQ1: What methods exist to reduce and control the cost of DL?
- RQ2: Which of these methods are available without developing, buying, distributing, or supporting any hardware?
- RQ3: What are the pros and cons of these methods?

The goal of RQ1 is to broadly analyze and categorize existing methods for cost-efficient DL. RQ2 scopes the review of methods available to organizations that cannot or do not want to deal with DL-specific hardware. RQ3 further analyses the methods that pass the criteria set in RQ2.

B. SEARCH STRATEGY

We conducted the search in three databases (IEEE Xplore, ACM Digital Library, and Scopus). We scoped the search timeline to years from 2010 to 2021 based on the observation that DNN research has re-activated during the 2010s. Fig. 2 summarizes the search and selection process.

First step, we prototyped search terms. Based on a preliminary search for papers, we found that only a handful of papers concentrate specifically on the *cost-efficiency* of DL. We had to consider what other goals might lead to the same result: saving cost. Saving energy, a popular topic in the past few years, involves methods for decreasing the amount of computing or its resources, which could also save cost in the context of DL. We found that including energy efficiency in the search criteria discovered many potentially relevant papers but also increased the manual work of going through a more extensive set of search results, for example, the number of papers matching the criteria from IEEE Xplore tripled. Consequently, we combined terms related to costs and energy with terms related to Artificial intelligence (AI) in our search terms as described for each scientific database as follows.

IEEE Xplore offers an API for programmatic queries [33]. We applied the search string to all metadata and limited the search to journals, magazines, and conference papers. We set the period between 2010 and 2021. The following query resulted in 9374 papers.

```
(artificial-intelligence OR AI OR
 deep-learning OR DNN OR
 machine-learning OR ML OR
 neural-network*)
AND
(cost-aware OR
 cost-eff* OR
 cost-reduction OR
 energy-aware OR
 energy-eff*)
```

We used the same search string and time range as Scopus but limited the search to the title, abstract, and keywords. This query resulted in 4692 papers.

ACM Digital Library search strings do not support wildcards in phrases. Consequently, we used the following search string for the ACM Digital Library:

```
("artificial intelligence OR "AI" OR
 "deep learning" OR "DNN" OR
 "machine learning" OR "ML" OR
 "neural network")
AND
("cost aware" OR
 "cost efficient" OR
 "cost reduction" OR
 "energy aware" OR
 "energy efficient")
```

We limited the search to research articles published between 2010 and 2021. The query resulted in 4372 papers, however, unfortunately, the ACM Digital Library limited the viewing of the search results to the first 2000. We sorted the results in descending order by citation count and took the top 2000 most cited papers.

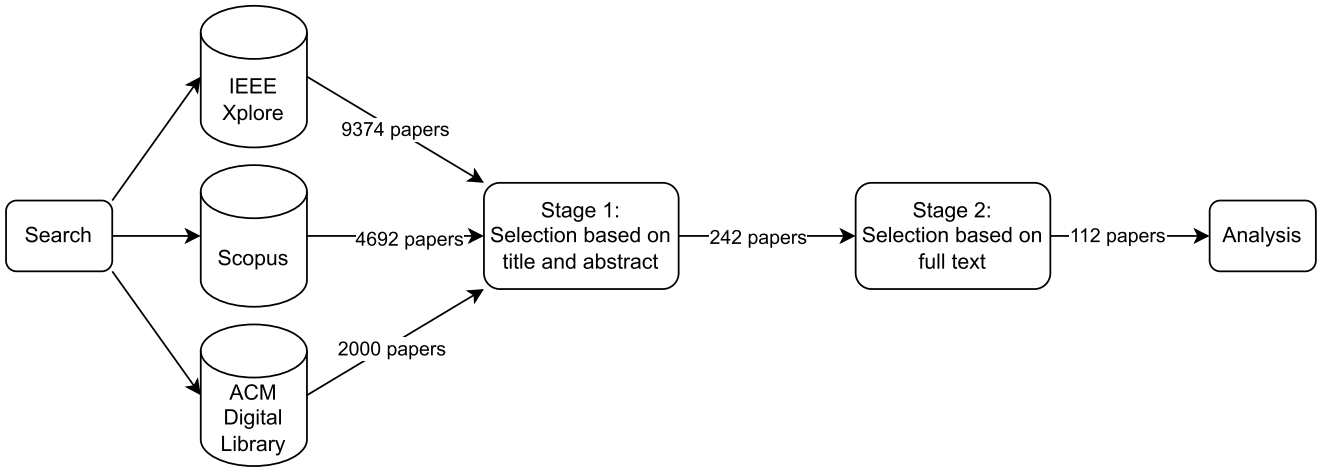

FIGURE 2. Search and selection process.

C. PAPER SELECTION

The combined set from the three databases included 16066 papers for which the inclusion and exclusion criteria were applied in two stages (cf. Fig. 2).

1) INCLUSION AND EXCLUSION CRITERIA

The papers had to fulfill both of the following criteria to be included:

IC1 The paper is about ML.
IC2 The paper is about optimizing some aspects of ML.

DL is a field of ML and DNNs are specific ML models. Because some optimization methods can be applied to ML in general and not only DNNs, the inclusion criteria cover ML.

The exclusion criteria were the following:

EC1 ML is used to optimize the energy consumption of some other domain, not ML itself.
EC2 ML is used to predict energy consumption in some other domains.
EC3 ML is used to optimize the cost of other domains, not ML itself.
EC4 ML is used to predict the cost in some other domains.
EC5 AI and ML abbreviations mean something other than Artificial Intelligence and Machine Learning.
EC6 Optimization applies only to other types of ML models than DNNs.
EC7 The paper is about a hardware accelerator without commercial implementations or significant academic interest (less than 100 citations).
EC8 There is no experimental evidence.

EC1–EC5 were necessary as expressing these criteria in the search string was difficult. We wanted to avoid enforcing EC6 with the search string as there are methods to improve the cost-efficiency of ML, which are generic to many types of models, including DNNs, such as data pre-processing by feature selection. Our research problem implies EC7 because most hardware methods are unavailable for software-only development organizations. EC8 excludes pure conceptual or solution proposal papers without evidence of how the proposed method works in practice.

2) STAGE 1: SELECTION BASED ON TITLE AND ABSTRACT

We manually analyzed the search results from all databases based on the titles of the papers and, when necessary, on abstracts. Duplicates found in more than one database were excluded at this stage.

3) STAGE 2: SELECTION BASED ON FULL TEXT

We analyzed the full papers, starting from the introduction and conclusions and drilling down to the rest of the text when necessary. The inclusion and exclusion criteria were the same as those for stage 1. We also outlined the initial categorization of papers for the analysis.

D. ANALYSIS

In the analysis stage, 112 papers were read thoroughly, and the final categorization for RQ1 was decided. The data extraction form is presented in Table 1. The categories emerged from the analysis. We decided to make the values for the categories mutually exclusive to make it easier for readers to follow. We discuss all the primary studies in Section V in dedicated subsections per category.

The quality of the included papers was assessed by assigning evidence levels to each paper. We decided to use the evidence levels [34] that we interpreted in our context as follows, from weakest to strongest.

L1 No evidence.
L2 Evidence obtained from academic experiments, but the datasets used are not mentioned (feasibility study) or provided a toy example.
L3 Evidence obtained from academic experiments using simple public datasets or datasets not mentioned, but the experimental setup otherwise described in detail.

TABLE 1. The data extraction form.

Extracted data	Values
Title	The title of the paper
Authors	The names of the authors
DOI	Document Object Identifier
Year	The publishing year of the paper
Category	Inference, Edge offloading, Hardware acceleration, Training, Hyper-parameter, Human effort (reduction), Feature selection
Subcategory	Pruning, Quantization, BNN, Early exit, Teacher-Student, Compact CNNs, (Inference) Usage patterns, Algorithm co-design, New numeric formats, Accelerator, Inference, Training, Both, (Hyper-parameter) Tuning, (Human effort) Reduction, Feature selection
Evidence level	Evidence level of the paper
Optimize	The optimization goal of the paper: *Compute* to reduce the computational cost, *Energy* for energy efficiency, and *Labor* to reduce the human effort related to developing ML systems
CNN	*yes*, if the optimizations suggested in the paper are CNN-specific, otherwise *no*
Public datasets used	The names of the public datasets used

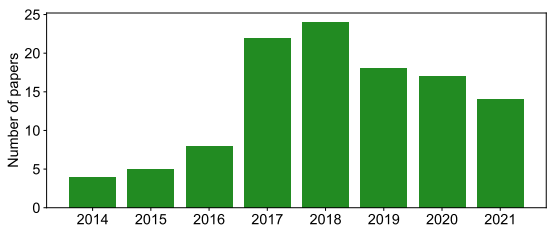

FIGURE 3. The number of papers by year of publication.

L4 Evidence obtained from academic experiments using complex public datasets and the experimental setup described in detail.
L5 Evidence obtained from industrial proof-of-concept
L6 Evidence obtained from industrial practice

Papers without evidence (L1) were excluded during the paper selection, as all included papers had experimental evidence (EC8). We interpreted L3 differently than [34]: L3 and L4 apply the complexity of the datasets used as indicators of stronger evidence.

CNNs play a significant role in research on making DL more efficient. Therefore, we separately mark the papers in which the optimizations suggested are based on CNN specific structures, for example, convolution or pooling layers. We also gathered the names of the public datasets used in the papers.

IV. OVERVIEW OF THE INCLUDED PAPERS

The references and tabulated data extraction results for the included primary studies are listed in Appendix A. The included studies are ordered alphabetically and referenced hereafter using the letter "S" as the prefix, as in [S1] for the first paper. The distribution of the papers by year of publication is shown in Fig. 3.

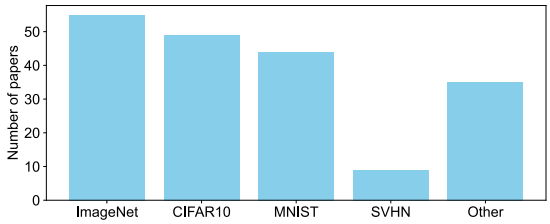

FIGURE 4. Public datasets used in papers.

All included papers present experimental validation for the method that is being suggested. A total of 100 out of 112 papers applied publicly available datasets for the experiments. The remaining 12 papers without public datasets mentioned have the following evidence levels:

- Six papers [S1], [S2], [S3], [S4], [S5], and [S6] have L2, which means that the experimental setup is not thoroughly described.
- Five papers [S7], [S8], [S9], [S10], and [S11] have L3, which means that the experimental setup is otherwise well described, but the dataset is not mentioned.
- One paper [S12] has L6, which means the results have been obtained via credible industrial practice.

Fig. 4 shows the distribution of public datasets used in these papers. The most commonly used datasets, that is, ImageNet, CIFAR10, MNIST, and SVHN, contain only images. All other public datasets were spread evenly with one or two uses. Most 'Other' datasets also are image datasets, and the rest are related to natural language processing, such as LibriSpeech.

Among the most commonly used datasets, ImageNet has the most samples and images with the highest resolution. It contains 1,281,167 training images, 50,000 validation images, and 100,000 test images. The images have 1,000 object classes, and the average resolution of images is 469×387 pixels. CIFAR10 consists of 60,000 32×32 resolution images that belong to ten classes. Street View House Numbers (SVHN) dataset has more than 600,000 32×32 resolution images that belong to ten classes. The MNIST dataset contains 70,000 28×28 resolution images that belong to ten classes. Appendix A presents the datasets used in each paper.

The bubble chart in Fig. 5 summarizes the distribution of papers by evidence levels in each subcategory, as described in Section V. The sizes of the bubbles indicate the number of papers. Both the median and mode of evidence levels are L4 indicating that most papers include experiments with public datasets that can help with repeatability.

Because we included energy as a keyword in our search, we collected the optimization goal for each paper (Optimize column in Table 1). Energy efficiency dominates the optimization goals appearing in 61% of the papers; reducing the computational cost is the goal in 36% of the papers, and 3% aims to reduce the human effort of developing ML systems.

V. COST-EFFICIENT DEEP LEARNING METHODS

We categorize the cost-efficient methods for DL into seven categories and 13 subcategories, as shown in Fig. 6. For

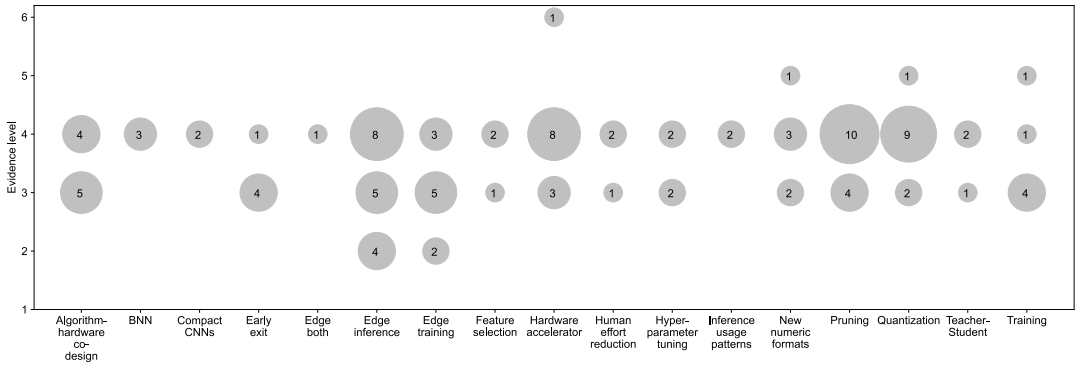

FIGURE 5. Papers by evidence level and category.

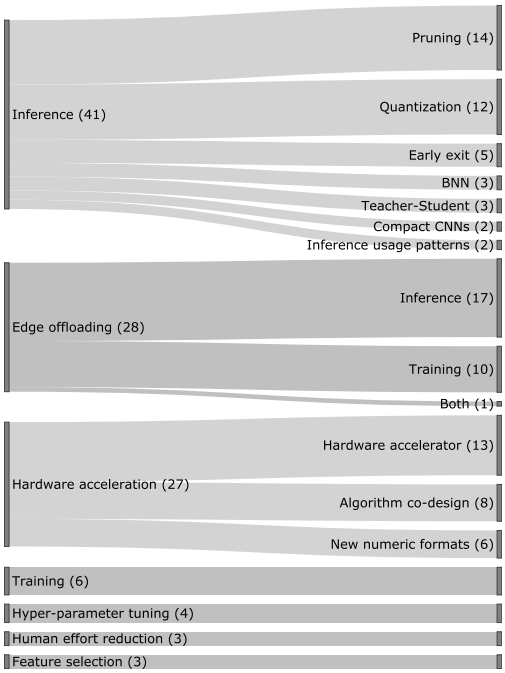

FIGURE 6. Included papers by categories.

each of these categories, we introduce the methods found through the literature search in the following sections. Our categorization represents two different approaches to methods. The first approach considers the optimization areas of ML model development: human effort reduction, feature selection, training, hyper-parameter tuning, and inference. The second approach examines the execution environment for ML: edge and cloud. Edge computing has different resource constraints than cloud computing, such as network bandwidth and processing power. The methods from the first perspective apply to both edge and cloud computing.

A. INFERENCE

37% of the papers are about making inference more efficient by reducing the model size and computational complexity. We divide inference optimizations into pruning, quantization, teacher-student method, early exit methods, binary neural networks, compact CNNs, and inference usage patterns.

1) PRUNING

Pruning means making DNNs smaller by removing connections or neurons that contribute the least to the accuracy of the model. DNNs are typically over-parameterized, and pruning has proven to be an effective way to reduce their size. The most straightforward approach is to prune those connections with weights close to zero, as the MAC operations of the neurons are least impacted by the weights close to zero. If all weights for all incoming connections in a neuron are close to zero, the entire neuron can be pruned. Often 80-90 percent of the parameters can be omitted without decreasing the accuracy [S13].

What should be pruned: Connections, neurons, layers, filters, or channels? Connection pruning is an unstructured pruning method that produces sparse matrices that require customized software compared to standard matrix operations. Rather than operating on contiguous memory areas, the indices of the connections remaining after pruning are used. Sparse matrices are often presented using the Compressed Sparse Row (CSR) format [35], which stores the indices of non-pruned weights. According to [S14], unstructured pruning should be discouraged because of the overhead of irregular memory access. Pruning neurons or layers produces dense matrices that can be computed using standard matrix operations with regular memory access. Unfortunately, neuron pruning can reduce the model size to a lesser extent than connection pruning [S15]. In the case of CNNs, filters or channels can be pruned, resulting in dense matrices and significant cost savings [36].

What are the pruning criteria? Instead of pruning weights close to zero, neurons producing similar outputs regardless of the inputs can be pruned [S16], [S17]. Principal Component Analysis (PCA) can be performed per DNN layer to find and prune the redundant neurons for each layer [S18]. Instead of layer by layer, pruning can be performed for the entire network using the average of absolute weights per neuron divided by the average of the layer [S17]. The energy

efficiency can also guide pruning. Pruning DNN parts that require the most frequent memory access and the heaviest computations is suggested in [S19]. For example, convolutional filters are accessed and applied multiple times during inference, making them more attractive targets for pruning than neurons in fully connected layers. Coarse-grained pruning looks at the blocks of a model and removes all related connections when a block meets the pruning criteria [S20]. The resulting matrices are sparse in a regular manner, which makes the hardware-accelerated inference efficient.

When should pruning occur: during or after the training? Pruning can be performed during training for each epoch [S21]. The insignificant weights in the early training epochs often remain insignificant until the end of training [S22]. Pruning while training implies keeping a shadow copy of the already pruned weights for recovery when facing more significant than acceptable accuracy loss. Reference [S23] proposes an incremental training and pruning approach. Connections are not only removed but are also restored during training. Pruning can occur after training, and usually involves an additional step of retraining the pruned model to restore some of the lost accuracy [S16], [S24]. Pruning can also occur just before inference. A CNN classifier pruned dynamically in the client device based on the relevant classes for that client is proposed in [S25].

How to measure pruning success? Pruning is often considered successful when the model accuracy does not decrease significantly, but according to [S26], pruning can impact the classification confidence scores more than pruning decreases the accuracy. This can lead to increased overall computations when confidence scores are extensively used in the further processing [S26].

2) QUANTIZATION

The DNN weights are typically represented as 32-bit floating-point numbers. Quantization means converting weights and sometimes activations to lower-precision integers. The conversion places continuous floating-point values into discrete integer buckets. In bucketing, the floating point values are divided by a scaling factor and rounded off to integers. The value of the scaling factor depends on the range of floating point values to be bucketed. The number of buckets depends on the bit-width of the integer type used. When using smaller bit-width for weights, the models become smaller, reducing off-chip memory access. Integers are also lighter to multiply and accumulate than floating point numbers, further reducing costs.

Quantization may decrease model accuracy by introducing rounding errors. As a solution, stochastic rounding during training has been proven to be effective [S27]. Stochastic rounding means that a decimal value is not always rounded up to the closest integer but rounded probabilistically, which leads to an unbiased result on average [S27]. The rounding errors are also balanced by their regularization effect, that is, rounding errors reduce over-fitting to the training set. This explains why quantized models sometimes have higher accuracy against unseen inputs than the original floating point models [37].

When to quantize during or after training? Quantization can occur after training, or training can be quantization-aware. Training without knowing that the model will be quantized for inference can dramatically decrease the model accuracy. Quantization-aware training can recover accuracy but may require changes in the model architecture [S28]. Quantization-aware training maintains two copies of weights: the full precision version and quantized version. Sometimes quantized weights are used in forward propagation, and floating point numbers are used in backward propagation [S29]. Training using 8-bit and 16-bit floating point numbers is possible without significant accuracy degradation when using stochastic rounding and chunk-based accumulation [S30]. Chunk-based accumulation prevents the overflow of 8-bit variables that store MAC operation results. The use of integers to approximate all numeric values during training, including inputs, weights, activations, gradients, and errors, is described in [S31].

Which bit-precision to use for quantization? 8-bit integers are a common choice for precision [S32], [S33], [S34]. 8-bit integers benefit from not requiring custom inference hardware but having a wide enough range to maintain model accuracy. Moderate accuracy loss has been achieved with 6-bit precision [S35], 4-bit precision [S36], and ternary values [S37]. Quantization further reduces the model accuracy for simpler models [38]. The more complex the model and the more neurons the model has, the less quantization decreases the accuracy. When using low precision – binary or ternary – more complex models compensate for the accuracy loss.

What is a good granularity level for bucketing? Bucketing can be performed for the entire model, by layer [S32] or by a CNN filter [S38]. Bucketing at the layer level can lead to fewer buckets, which requires smaller integer bit-width, which reduces the model size without significant accuracy loss [S32].

What to quantize: weights or activations? When quantizing the weights, the inference load is lowered. On the one hand, quantizing activations, including inputs, leads to inference-time bucketing, which adds computations. On the other hand, quantizing activations during inference can still be lighter than using floating point numbers for activations [S34].

3) BINARY NEURAL NETWORKS

Binary neural networks (BNN) take the compression of weights and activations further by representing them with just one bit [S39]. Compared to 32-bit floating point DNNs, BNNs mean 32 times smaller models and radically reduced off-chip memory access during inference. Computations become light-weight as multiplications can be replaced by bit-wise operations, such as XNOR [39]. BNNs are promising for low latency and low energy inference, although optimal

BNN execution benefits from custom hardware. Another significant drawback of BNNs is the reduced model accuracy. One suggested approach to increasing the BNN model accuracy is to combine ensemble methods with BNNs [S40].

4) TEACHER–STUDENT

One way to lower the inference resource usage is to use simpler models trained using deeper and more complex models. The concept of complex teacher models and simpler student models is introduced in [40]. The student model is trained with numeric predictions made by the teacher model instead of using the labels. In other words, instead of using the results from the teacher's softmax function, the student is trained using the logits that are the inputs for the softmax function typically used in classifiers. Both logits and labels are used in [41], where the method is named knowledge distillation. The teacher and student can have a completely different DNN architecture. However, imitating not only the logits and labels but also the architecture of the teacher with fewer layers appears to achieve high accuracy [S41]. The teacher and student can even have the same architecture, and the result can be a lighter student model if the student is trained using quantization or ternarization [S42]. In the case of a distributed inference, multiple student models can be trained from a teacher model, and each student concentrates on a set of distinct CNN filters of the teacher [S43].

5) EARLY EXIT METHODS

Not all model inputs are equally difficult and, therefore, costly to classify. However, the same, possibly complex model is typically used for all inputs. To reduce the cost of handling simple inputs, a chain of classifiers with increasing complexity can be trained [S44]. Each classifier in the chain outputs a confidence score for its prediction, which determines whether a more complex classifier needs to be evaluated. When a simple model produces a prediction with sufficiently high confidence, the more complex models in the chain do not need to be evaluated and hence, the inference exists early. In addition, the consensus between classifiers that have already been evaluated can be verified [S44]. If there is no consensus despite a high confidence score, a more complex classifier must be applied. Choosing the threshold values for early exits is a balance between computation cost and prediction accuracy. This problem is formalized in [S45].

The first layers in CNNs are convolutional, and act as feature extractors. These first CNN layers can be the same for all classifiers with different complexities [S46]. To reduce the computational cost, more complex models can reuse the inference computations from the layers shared with simpler models [S47]. BNNs can also be used as the first light-weight step in early exit inference [S48].

Instead of training each model separately, the training epochs of the models can be synchronized, and the loss from all the exit points can be used jointly in backward propagation [S49]. In other words, the training samples are passed through all exit points in forward propagation, and the errors from all exit points are fed to a joined loss function from which the resulting loss is used in backward propagation for all models. This increases the accuracy of the less complex models.

6) COMPACT CNNs

Since the introduction of AlexNet [6] in 2012, CNN research has focused on balancing energy efficiency, inference latency, and accuracy. The trend has been to create CNN architectures with fewer parameters and minimal accuracy loss: MobileNet is a compact CNN architecture based on depth-wise separable convolutions [42]. SqueezeNext is a CNN that uses bottleneck layers to decrease the number of parameters [S50]. SqueezeDet is a CNN for object detection with autonomous driving in mind [S51]. SqueezeDet has only convolutional layers and no fully connected layers, which results in a small model size and heavy memory reuse.

7) INFERENCE USAGE PATTERNS

Not all inference inputs and workloads are similar, and inference usage patterns can be used to reduce inference costs. Consecutive inputs are often very similar when processing streams of images or audio. By saving the intermediate results of the previous inference, the results can be reused in the following inference when the input is sufficiently similar [S52]. Another approach builds on the fact that calls to inference services can be bursty, and allocating a fixed number of cloud instances or even using auto-scaling of cloud instances is not cost-efficient. Instead, serverless computing, such as AWS Lambda, and light-weight cloud instances for batching and buffering can be more cost-efficient for bursty workloads [S53].

B. FEATURE SELECTION

Selecting the most predictive input features reduces model complexity, computational resources required, and cost. Feature selection can be integrated into the model development process in three different ways [43]: First, filter methods, such as K-means clustering, hierarchical clustering, and principal component analysis (PCA), use unlabeled data to reduce the number of features before model training based on feature correlations. Second, wrapper methods, such as linear discriminant analysis (LDA), train several models with different combinations of features and select the best model with a given goal, such as accuracy or cost. Third, embedded methods integrate feature selection into the model training. Embedded methods attempt different sets of input features during training, estimate their impact on the loss function, and select the features with the most significant impact.

Wrapper and embedded methods can be too costly, and filtering methods can significantly reduce model accuracy [S54]. Semi-supervised feature selection can be used as a solution [S54]. In semi-supervised feature selection, a small fraction of the data is labeled, which leads to a higher model

accuracy with fewer input features than using only unlabelled data for the selection.

The cost of data collection is not the same for all input features; however, the selected features can be a compromise between the cost of data collection and model accuracy [S55], [S56]. For instance, different medical tests for collecting input data have different costs, such as blood tests cost less than magnetic resonance imaging.

C. TRAINING

The methods described in Section V-A aim for more efficient inference execution in resource-constrained devices. More efficient inference often comes at the cost of extra processing during training. This, combined with the ever-increasing amount of training data, makes training a target for optimization.

Graphics Processing Units (GPU) are optimized for the parallel execution of high-dimensional matrix operations involved in DNNs. GPUs are commonly used to accelerate DNN training, but DNN models have grown so large that it is impossible to train them using GPU's own memory. In training, both weights and activations must be kept in memory between forward and backward propagation. Instead of keeping all the data all the time in GPU memory, studies [S57], [S58] offload the layers not actively used to CPU memory which introduces a moderate performance penalty.

Distributed training can shorten the training time; however, it is typically more costly than using one machine beacuse of data synchronization over the network. Distributed training can be the only option when the size of the input data and the trained model is sufficiently large. The cost of distributed training can be reduced using transient or volatile cloud instances [S59].

Regardless of the approach (training using one or more machines), there are ways to reduce the training cost by less or lighter computations. Random dropout of neurons in each training epoch is a commonly used technique to reduce over-fitting [S60]. Instead of random dropout, an adaptive dropout can be applied based on the values of activations where only the top k neurons are executed and updated on each epoch [S60]. The gradient calculation is one of the heaviest computations required for training. For CNNs, gradients can be approximated instead of calculating all of them [S61]. Convolution filters dominate the computations for both the CNN inference and CNN training. Computationally less heavy Gabor filters can replace convolutional kernels [S62].

D. HYPER-PARAMETER TUNING

Determining the optimal hyper-parameters is a multi-objective optimization problem that requires engineering effort and computational power. Several hyper-parameters, such as the DNN architecture, regularization method, and learning rate, can be optimized. There are also multiple objectives for optimization, such as model accuracy, training time, training cost, inference cost, inference latency, and energy consumption. The energy consumption of different DNN structures, for example, convolutions, can be estimated based on their ability to reuse the on-chip memory [S9]. Based on these estimates, the DNN architecture can be tuned toward an energy-efficient inference.

A simple grid search over hyper-parameter values can be costly. A more efficient algorithm for DNN architecture search is proposed in [S63]. The algorithm takes the initial DNN architecture, description of the target hardware, inference latency threshold, and inference energy consumption limit as the input and outputs an optimized DNN architecture. Reinforcement learning is used for the same purpose as that in [S64].

Hyper-parameter tuning requires computational resources in addition to training and inference. Transient or volatile cloud instances, such as AWS spot instances or Google Cloud's preemptible instances, can be used for hyper-parameter tuning [S65]. These cloud instances are significantly cheaper than on-demand instances but require a persistence strategy and orchestration when training is interrupted by instance revocation.

E. EDGE OFFLOADING

The cost of cloud computing can be reduced by offloading computations to the network edge. Of the included primary studies, 26% are on edge offloading.

1) TRAINING AT THE EDGE/FEDERATED LEARNING

Distributed training at the edge, also known as federated learning, means that there is no single centralized training dataset but many participants improve a global model from local datasets and share the updated model. Training using client devices can reduce cloud costs, although cloud network ingress and egress also contribute to the cost. To make distributed training in client devices cost-efficient, the increased network traffic cost cannot exceed the saved cloud processing cost. Despite its potential for cost saving, the most frequently mentioned motivation for federated learning is data privacy [S1], [S4], [S66], [S67], [S68]. The model inputs, which may contain user privacy-related fields, do not need to be stored in the cloud, only the resulting models.

Battery consumption may become an issue when training occurs on mobile devices. For example, GPU usage in a mobile device can rapidly drain the battery. Synchronizing model parameters and training data over a network also consumes battery. Generally, training must not hog the limited resources in a mobile device; otherwise, the device can become unusable for running end-user-facing applications. Distributed training in mobile devices is a balance between the local processing needed, network transfer needed, required model accuracy, and the number of iterations for the model to converge.

There are two primary ways to split the work in distributed training: first, data parallelism, in which all weights are shared between all participants, and each participant

processes a separate set of samples. The second is model parallelism, in which the weights are partitioned between participants, and each participant uses only selected fields from all the training samples. Data parallelism is typically preferred in federated learning However, in the case of CNNs, a hybrid version of data and model parallelism seems optimal [S69]. The convolutional layers have significantly fewer weights than the fully connected layers. Thus, data parallelism is a better choice for convolution layers, whereas model parallelism is more efficient for fully connected layers.

Federated learning involves finding a balance between "working" (local processing) and "talking" (network transmission) [S4], [S70]. The network bandwidth can be saved by sparse weight updates where only the updated weights are communicated to other participants [S70], [S71]. The frequency of sharing the updated weights between the training iterations is a tunable hyper-parameter [S70], [S71], [S72]. If the weight updates are frequent, the shared model may converge faster; however, the network latency and related energy consumption may increase. Model accuracy can also be sacrificed if faster and more efficient training is preferred [S4]. The updated weights can be communicated directly between client devices or via shared parameter servers. MEC edge servers can be used as aggregators and distributors for models [S67], [S68].

The participants in federated learning may have heterogeneous hardware, and the processing load between participants at a given time may differ. This information can be used to determine how to distribute the training tasks among participants [S1], [S66], [S73].

2) INFERENCE AT THE EDGE

Running inference in the cloud has drawbacks that inference at the edge tries to avoid:

- User privacy, e.g., the input contains sensitive data that are not sent to the cloud [S74], [S75], [S76], [S77], [S78]. Not sending user privacy-related data to the cloud and not processing the data in the cloud also reduces the cost.
- Latency, e.g., the round trip to the cloud data center and back is long [S6], [S8], [S10], [S74], [S78], [S79], [S80], [S81].
- Communication cost, e.g., the energy consumption of the data transfer from the client device [S3], [S8], [S74], [S76], [S77], [S81].
- The monetary cost of cloud computing, e.g., the cost of running the models and the cloud network [S10], [S79].
- Dependence on network connections, e.g., client devices require an always-on network connection [S75].

Despite its benefits, inference at the edge is problematic. Client devices have heterogeneous hardware [4], and the processing power is not comparable to the hardware available in the cloud. Many client devices, such as mobile phones operate on a batteries with limited lifetimes.

Three primary streams of research have attempted to enable client-side inference:

1) Develop smaller DNNs using the methods described in V-A, for example, quantization or pruning.
2) Create tooling or adaptation for efficient inference execution in resource-constrained devices. These tools include DL libraries or frameworks [44], [45] and compilers that map high-level DNN descriptions to efficient low-level executables [S5], [S74], [S75], [S79], [S82].
3) Split the inference execution between the client device and edge server layer or cloud [S2], [S3], [S6], [S8], [S10], [S43], [S76], [S78], [S80], [S81], [S83], [S84].

The above approaches complement each other.

The splitting of inference execution deserves a closer examination here. The split can occur by running some DNN layers on the client device and the rest in the cloud or edge server [S8], [S3], [S80], [S81]. If edge servers are available, the layer partitioning can be driven by energy consumption [S2], [S6], or cost [S10].

A typical CNN architecture hints at where to split the inference execution. The first CNN layers are typically convolutional, and the last layers are fully connected. The number of weights in the convolutional layers tends to be smaller than that in the fully connected layers, but the processing required per weight is much heavier in the convolutional layers than in the fully connected layers. The pooling layers typically follow convolutions to reduce the dimensions of the inputs for the following layers. If the network connection and memory available in the client device are limited, but there is decent processing power, executing the convolution and pooling layers in the client device and the fully connected layers in the cloud make sense [S80]. This split also supports the user privacy aspect of not sending raw inputs outside the client device. However, if the inputs are small, for example, typical of GANs, the opposite execution plan makes sense [S8].

Splitting the inference execution can also occur by training two or more separate DNNs: light-weight models for the client device and heavier models for the edge or cloud server. The heavier models are executed if there is not enough confidence in the prediction produced by the light-weight models [S76], [S81], [S83], [S84]. The early exit approach is discussed in Section V-A5. The conditional execution of heavier models can reduce the latency and cost in the average case but cannot improve the worst case.

A different way of looking at inference and client devices is to consider them as a cluster of workers. The workers can be IoT devices with non-overlapping inputs for inference, and each performs a partial inference on the local data [S77]. Alternatively, client devices in the same wireless network can share the same input and perform cooperative inference [S78]. In cooperative inference, each client has its own student model trained to handle a particular set of CNN filters from a larger teacher model [S43].

3) TRAINING AND INFERENCE AT THE EDGE

One primary study covered both training and inference at the edge as a whole, suggesting that DNN layers can be split between clients and servers in training and inference [S85].

F. HARDWARE ACCELERATION

Most publications related to cost or energy-efficient DL concentrate on hardware acceleration. DNN execution involves intensive memory access and parallelism, which are not supported by general-purpose processors [46]. We excluded 976 papers related to ML-specific hardware from the IEEE Xplore search results, 676 papers from Scopus search results, and 178 papers from the ACM Digital Library search results. We included papers only on hardware accelerators that are already commercially available, for example, TPUs and FPGAs, and highly cited papers, which might lay the foundations for new commercially available accelerators. We included papers on algorithms and new numeric formats for more efficient DNN inference, which are interesting innovations, although specialized hardware is needed.

1) ALGORITHMS FOR INFERENCE ACCELERATORS

DNN inference is a sequence of matrix operations, or in more general terms, tensor operations. When the DNN-specific properties of these operations are known, both the computations and memory access can be reduced. However, inference is no longer a straightforward set of matrix operations. All methods in this section work efficiently only with custom algorithm-aware hardware.

ReLU is a commonly used activation function in DNNs. ReLU maps all negative inputs to zero. Multiplying zero valued activations with weights is useless because the result is also zero. The sign of an MAC operation can be predicted using high-order bits of input activations and weights [S86]. If the sign of the MAC operation is predicted to be negative, the result of ReLU will be zero, and there is no need to perform the MAC operation or ReLU.

Smaller matrices are more likely to fit into the on-chip memory, which makes them faster to compute. Low-rank matrix decomposition can express a large weight matrix as the product of two smaller weight matrices [S87]. The weights can be converted to block-circulant matrices that enable a Fast Fourier Transform for efficient multiplication during inference [S88], [S89], [S90]. The Fast Fourier Transform reduces the computational complexity from $O(n^2)$ to $O(n \log n)$.

Neurons can be classified as either sensitive or resilient. Resilient neurons can be discovered during training and replaced with light-weight approximations during inference [S91]. Bloom filters can store the frequently occurring activation patterns in cache for inference calculations [S92]. The size of the model can be reduced through weight sharing. Weights are grouped using K-means clustering, and the weight matrix is replaced by a lookup table [S93]. Similar lookup tables are used for sparse matrices resulting from unstructured pruning (see Section V-A1). Efficient lookup tables can be implemented by using custom hardware.

2) NEW NUMERIC FORMATS

Quantization (see Section V-A2) may reduce the bit-width of weights to something that is not well supported by general-purpose hardware. Floating point numbers, according to the IEEE 754 standard, are computationally heavy, and the 32-bit format consumes much memory compared to its contribution to model accuracy. Instead of approximating 32-bit floats using integers, new numerical formats have been proposed for representing real numbers.

The Posits numbering approach has been suggested to replace floating point numbers with better accuracy but using fewer bits to represent the numbers [S94]. According to [S95], by using Posits instead of fixed-point numbers, the same inference accuracy can be achieved with fewer bits. Training DNNs with 8-bit Posits without accuracy loss compared to using 32-bit floating-point numbers is presented in [S96]. Using Posits efficiently requires dedicated hardware support that is yet to be widely available.

Flexpoint [S97], another numerical format, was suggested approximately the same time as Posits. Flexpoint allows the sharing of the exponent part, and magnitude of numbers separately for tensors used as weights, activations, and updates. Less memory is consumed, and the computations can be mostly done using integers. To operate efficiently, Flexpoint requires custom hardware. Perhaps because it is specific to DNNs, Flexpoint has spawned less further research than Posits.

"Binary floating-point" representation for model weights has been suggested in [S98]. When using 4-bit and 6-bit precision, there is no accuracy loss in the benchmarks. A new number format that allows the storage of information about quantization-related rounding errors, for example, the direction and magnitude, is proposed in [S99]. The direction and magnitude fields enable the recovery of some of the prediction accuracy lost in quantized inference, but custom hardware is required.

3) HARDWARE ACCELERATORS

GPUs are widely used in accelerating DNN training, however, they consume a significant amount of energy. GPU-enabled instances are available from all the major cloud providers for both training and inference. The high energy consumption of GPUs has led to a wide variety of studies aimed at accelerating DNN inference with less energy-hungry hardware. Alternatives to GPUs are Application-Specific Integrated Circuits (ASIC) and FPGAs.

The most widely used ASIC implementation for DNNs is TPU [S12]. TPUs originated from Google, and it is no surprise that Google Cloud Platform's inference acceleration is based on TPU-enabled instances. In addition to TPUs, many conceptual ASIC designs that minimize off-chip memory access and take advantage of weight and input sparsity have

been proposed [S7], [S100], [S101], [S102], [S103], [S104]. These designs have gained broad interest in academia but lack commercial implementations in the cloud and mobile platforms.

Amazon and Microsoft have taken the path of offering FPGA-enabled cloud instances for DNN inference [20]. FPGA implementations provide speedups against CPUs and on-par performance with GPUs but outperform both CPUs and GPUs in energy efficiency [S105], [S106], [S107]. The convolution loops in CNNs enable parallelism, and on-chip memory reuse [S11], [S108], [S109] for FPGAs. The optimal mapping of DNNs to cloud FPGAs require extra effort compared with simply using GPU- or TPU-enabled cloud instances via a deep learning library, and the same people are rarely experts in both DNNs and FPGAs. Automated tooling that maps high-level DNN descriptions to optimal FPGA implementations is required [S106].

Instead of attempting to map DNN execution to the current hardware or developing hardware that matches the needs of DNNs, attempts have been made to mimic how the human brain works, referred to as neuromorphic hardware or neuromorphic computing. An overview of neuromorphic computing as an alternative to DNNs is given in [47]. All papers on neuromorphic computing caught that met our search criteria were excluded. Neuromorphic computing is not yet an option for software organizations that rely on the existing platforms.

G. REDUCING THE HUMAN EFFORT RELATED TO ML

With large-scale production usage of ML, the majority of the costs are often related to computations in the cloud. In some cases, the cost of human effort may be higher than that of cloud usage. This can be the case, especially for startup companies in the early phases of their journey. The most considerable human effort is related to model development and training samples labeling.

Labeling data for supervised learning requires human effort. Smart selection of the samples to be labeled is referred to as active learning or active labeling. The following DNN classifier metrics can be used for active labeling [S110].

- Least confidence. Label the samples with the lowest confidence score.
- Margin sampling. Label the samples with the smallest separation between the top two class predictions.
- Entropy. Label the samples with the highest class prediction information entropy.

On one hand, samples with a low confidence score should be labeled by humans. On the other hand, samples with a high confidence score can be automatically "pseudo-labeled" and are used for training a more accurate model without knowing the ground truth labels [S111].

Model development involves determining an optimal ML model architecture and other hyper-parameters. In transfer learning, part of the model, including the trained weights, is taken from an existing model. A trained model is transferred to a new but somewhat similar domain, and the model is retrained in that domain. The weights of the first layers are frozen, and additional domain-specific layers are trained using data from the new domain. This has proven to work particularly well with CNNs, where the first layers act as generic feature extractors, for example, detect vertical and horizontal lines. Active learning can be combined with transfer learning, and the sample selection can be based on the following measures [S112]:

- Distinctiveness. The uniqueness of the sample is calculated for the unlabeled samples with respect to the original DNN. More distinctive samples tend to take a unique path of activations during inference.
- Uncertainty. The confidence scores of the unlabeled samples from the original DNN are used for transfer learning.

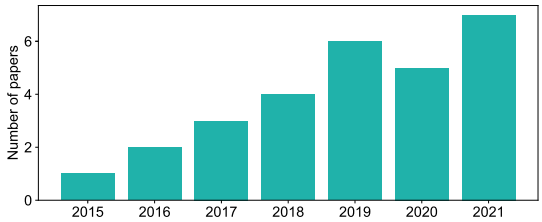

FIGURE 7. Edge offloading papers by year.

VI. DISCUSSION

This section presents our answers to the research questions and discusses our observations from primary studies.

A. ANSWER TO RQ1: METHODS FOR COST-EFFICIENT DL

The most popular methods are related to inference, edge offloading, and hardware acceleration, whereas the others are less popular. We assessed the quality of the papers based on their evidence levels (see Section III-D). 91% of the papers are on evidence level L3 or L4, meaning that most papers have repeatable experiments with public datasets. However, most methods have scarce industrial evidence, as only 4% of the papers are on levels L5 and L6, which require industrial adoption.

37% of the primary studies are about making inference more efficient. We divided the methods into pruning, quantization, teacher-student, early exit methods, BNNs, compact CNNs, and optimizations based on inference usage patterns. In industrial systems, the inference is typically executed more frequently than training [5]. For rapidly changing data, a new model can be trained daily. In comparison, for online prediction systems, the inference can be executed thousands of times per second with near real-time latency. These characteristics of inference might explain and justify why such a significant share of research is about inference optimizations.

25% of the primary studies focus on offloading inference, training, or both to the network edge.

The number of edge offloading papers shows an increasing trend from 2015 to 2021 (see Fig. 7). At the same time, the

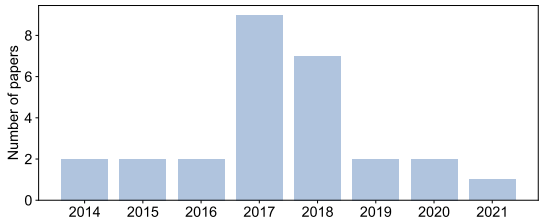

FIGURE 8. Hardware acceleration papers by year.

number of other DL optimization papers has been decreasing: Fig. 3 shows a downward trend in yearly published papers after 2018. All inference optimization methods in Section V-A were introduced before 2019. In 2017 Google presented the TPU accelerator [9]. We assume that TPU is viewed by many as a good enough commercially available solution for accelerating DNNs. This has probably reduced the interest in research on hardware accelerators (see Fig. 8).

24% of the primary studies focus on hardware acceleration. However, the percentage does not represent the actual share of research on hardware acceleration in terms of cost efficiency. In fact, almost 1900 papers from the search results were about efficient ML by hardware acceleration. Most were excluded owing to RQ2 and EC7, which assume that the hardware is commercially available.

5% of the primary studies focus on optimizing training. This surprisingly small percentage could be explained by the more frequent execution of inference, as stated above. Another possible explanation is that our literature search used energy efficiency as the criterion. Energy efficiency is less critical for server-side processing than for processing in client devices, particularly mobile devices. Training is typically performed server-side, in cloud.

The remaining papers were classified into three categories. Feature selection appears to be overlooked in the literature as a method for lowering the cost of DL, perhaps partly because of the heavy focus on image inputs. Intuitively, using the most predictive features and omitting the less predictive ones should reduce the model size, and thus, the cost. The rest of the primary studies focus on hyper-parameter tuning and reducing the human effort related to ML.

B. ANSWER TO RQ2: WHICH OF THESE METHODS ARE AVAILABLE WITHOUT DEVELOPING, BUYING, DISTRIBUTING, AND SUPPORTING ANY HARDWARE?

GPUs are available on all major cloud platforms, and TPUs are available on the Google Cloud Platform. The counterparts for TPUs in AWS and Azure clouds are FPGAs that require more configuration than TPUs for efficient DNN execution. It should be noted that using GPUs for training or inference does not necessarily lead to cost savings, but rather to faster execution. Using TPUs or FPGAs can lead to faster execution and reduced costs compared to CPUs. Both dominant mobile platforms, iOS and Android, provide ML APIs [48], [49], but in practice, mobile devices have heterogeneous hardware [4]. Some, but not all, mobile devices are equipped with hardware accelerators.

BNNs and, more generally, all quantization approaches that use bit widths of less than 8 bits require accelerator hardware adjusted for the lower bit width. The same applies to new numeric formats. All the other methods presented in this study are applicable without specialized hardware.

C. ANSWER TO RQ3: WHAT ARE THE PROS AND CONS OF THESE METHODS?

Pruning and quantization have been implemented in popular DL libraries, for example, TensorFlow [50] and Pytorch [51]. TensorFlow also supports the Teacher-Student method. However, none of these methods guarantees a zero accuracy drop. Notably, all related primary studies use accuracy as the evaluation metric. When using these methods, other ML model metrics should also be checked, such as precision, recall, and confidence scores of classification. In multi-class classification, it is worth checking if some classes suffer more than others with the compacted model. Nevertheless, pruning, quantization, and teacher-student are the first methods to consider when trying to reduce the cost of DL inference if a minor drop in accuracy is acceptable.

BNNs are very promising for lowering computational costs. BNNs cause unavoidable accuracy degradation unless the original model is highly complex [38]. The efficient implementation of BNNs also benefits from specialized hardware.

Early exit methods can be implemented without hardware acceleration and can lower the average case inference cost. The downsides of early exit methods are the increased system complexity, the new hyper-parameters for confidence score thresholds to be tuned, and the inability to lower the cost of the worst-case inference.

Interest in edge offloading has been increasing in academia, but the evidence level of the research has yet to reach the levels of other methods in this paper; there are no industrial primary studies, and the median evidence level is lower than for other primary studies (L3 for edge and L4 for others). In practice, the edge server layer is realized slowly. The reason could be the lack of a business or hosting model, the slowness of standardization, or the lack of "killer" applications that would benefit from the lower latency compared to the cloud. In fact, it has been demonstrated that the latency from client devices to the cloud in many parts of the world is sufficient for most applications [52].

The edge server layer cannot provide elastic "unlimited" scaling of the cloud. When an edge data center runs out of resources, client devices must have a fallback plan. Fallback processing can occur in cloud or resource-constrained client devices. Computation in edge servers can also incur a higher monetary cost than computation in the cloud [S10].

Although the future of the edge server layer seems uncertain, we see potential cost savings by offloading DL

processing to client devices. Even partial inference on client devices can save cost compared to cloud-only inference.

Only one primary study has examined training and inference at the edge as a whole. We claim that training and inference should always be viewed as a whole regardless of where they are executed. One of the most frequently mentioned benefits of edge inference is user privacy, as limited data must be sent to the cloud. By intuition, the same privacy restrictions apply to the training data, which is ignored by most edge inference papers that seem to assume the existence of training data without considering where it comes from. In these studies, the models are trained from public image datasets only once and then used as-is in the inference. In practice, concept drift often occurs [11], and new input samples are constantly gathered and labeled from the inference, after which the model must be retrained.

In primary studies on federated learning or training at the edge, the input samples are mostly images [S66], [S67], [S69], [S70], [S71], [S72]. Transferring images to the cloud requires more network bandwidth than transferring numeric readings from IoT device sensors. In both cases, the trained models can be large. For other than image inputs, new input features can be added over time, and the model architecture is updated. How can these updates be handled gracefully in federated learning? If the input data remains in the client devices in federated learning, how do the new input samples get the labels? The orchestration and aggregation of models trained using federated learning also seem non-trivial to implement. Instead, using a ready-made federated learning framework for the purpose makes sense.

D. FUTURE WORK

Image inputs dominate the public datasets used in the primary studies leaving research on other types of data thin on the ground. 78% of all the papers use ImageNet, CIFAR10, MNIST, or SVHN, which are image datasets. CNNs are mainly used for image inputs. 34% of the papers base the optimizations on CNN-specific structures, e.g., the convolutional and pooling layers. However, in 2017 in Google's data centers, CNNs were related to just 5% of DNN workflows [9]. Although images as input are typically larger than structural or tabular data and as such an interesting research area, we claim that further research on other types of data than images is needed to make DNNs other than CNNs more efficient.

A direction for our future work is to evaluate the methods presented in this paper with input types other than images using DNNs that are not CNNs. Specifically, splitting inference execution between edge devices and the cloud has been studied for image inputs [S80]. We see opportunities for splitting the inference execution for inputs other than images.

The combinations of inference optimization methods have been covered in some primary studies. First, the combined impact of pruning and quantization is measured in [35]. Second, quantization of the student model is proposed in [S42]. Third, the use of a BNN as the first model for an early exit approach is studied in [S48]. Other possible combinations of inference methods should also be evaluated.

None of the primary studies covered the optimization of GNNs, although modeling graph connections adds computational complexity and increases the memory footprint compared to MLPs. In the case of GANs, two models must be trained instead of one, but the search did not yield any studies on optimizing the execution of GANs. There seems to be a research gap in the area of optimizing GNNs and GANs, although certain inference optimization methods developed for MLPs also apply to them.

Although backward propagation and gradient descent are more resource-consuming procedures than forward propagation, our search found only one primary study related to improving the efficiency of computing the gradients [S61]. Our search did not match any primary studies that compare the cost impact of the gradient descent algorithms such as Adam [53], [54], although efficient gradient descent algorithms can significantly speed up the convergence [55] and reduce the cost of training. The choice of activation function impacts the cost of both inference and training; for example ReLU is cheaper to compute than Tanh, but our search did not find any primary studies on the topic. The cost impact of different gradient descent algorithms and activation functions should be evaluated in future studies.

The primary studies we reviewed concentrate on optimizing DL models. Modern industrial systems consist of a multitude of micro-services with extensive data transfer between them. Intuitively, there is a significant cost involved in data transfer. For example, in the case of structural or tabular input data, some inputs can be strings that must be converted to numeric values at the latest during model training and inference. The unified and compact input data types used across the system may result in even more significant cost savings than making the actual models more efficient to process. We consider this as a potential future research topic.

VII. THREATS TO VALIDITY

A systematic literature review should address explicit inclusion and exclusion criteria, cover a sufficient number of data sources, assess the quality of primary studies, and describe the data adequately [14]. We fulfill these criteria, except that we performed the search using three databases. Adding more databases or applying additional strategies, such as snowballing, would have provided better coverage, but the number of original and included papers was already relatively large (16066 and 112, respectively). Consequently, the analysis might have missed some studies, but we assume that we have reached a relatively valid sample.

The search strings try to find papers that are optimize the cost or energy consumption of ML. However, this does not rule out the possibility that we missed some other ML optimization goals.

Another limitation of the search was that the ACM Digital Library has a policy of limiting the number of search results to 2000. We selected 2000 papers that had the most citations.

TABLE 2. Information extracted from papers. The summaries of information are represented as follows: The publication years in Fig.3, and the publication years for edge offloading and hardware acceleration cateogries in Fig.7 and Fig.8, respectively; by category and subcategory in Fig.6; by evidence level and category in Fig.5; and public datasets used in Fig.4. CNN appears in 34% of the papers (cf. Section VI-D) and energy, compute, and labor values of the Optimize column represent 61%, 36%, and 3%, respectively (cf. Section IV).

Primary study	Year	Category	Subcategory	Evidence level	Optimize	CNN	Public datasets used
[S1]	2021	Edge offloading	Training	L2	Energy	no	
[S2]	2020	Edge offloading	Inference	L2	Energy	no	
[S3]	2018	Edge offloading	Inference	L2	Energy	yes	
[S4]	2019	Edge offloading	Training	L2	Energy	no	
[S5]	2020	Edge offloading	Inference	L2	Energy	no	
[S6]	2021	Edge offloading	Inference	L2	Energy	no	
[S7]	2015	Hardware acceleration	Accelerator	L3	Energy	yes	
[S8]	2018	Edge offloading	Inference	L3	Energy	no	
[S9]	2021	Hyper-parameter	Tuning	L3	Energy	yes	
[S10]	2020	Edge offloading	Inference	L3	Compute	no	Other
[S11]	2015	Hardware acceleration	Accelerator	L3	Energy	yes	
[S12]	2017	Hardware acceleration	Accelerator	L6	Energy	no	
[S13]	2018	Inference	Pruning	L3	Energy	yes	CIFAR10, MNIST
[S14]	2021	Inference	Pruning	L4	Compute	no	CIFAR10, ImageNet, MNIST
[S15]	2018	Inference	Pruning	L4	Energy	no	ImageNet
[S16]	2014	Inference	Pruning	L4	Compute	no	TIMIT
[S17]	2017	Inference	Pruning	L3	Compute	yes	CIFAR10
[S18]	2021	Inference	Pruning	L4	Compute	no	ImageNet, MNIST, WMT16
[S19]	2017	Inference	Pruning	L4	Energy	yes	ImageNet
[S20]	2018	Inference	Pruning	L4	Energy	yes	Other
[S21]	2020	Inference	Pruning	L3	Compute	yes	CIFAR10
[S22]	2019	Inference	Pruning	L4	Energy	no	CIFAR10, ImageNet, MNIST, TIMIT
[S23]	2020	Inference	Pruning	L4	Compute	no	ImageNet, MNIST
[S24]	2019	Inference	Pruning	L4	Compute	yes	CIFAR10, ImageNet, MNIST
[S25]	2017	Inference	Pruning	L3	Energy	yes	CIFAR10
[S26]	2018	Inference	Pruning	L4	Energy	no	LibriSpeech
[S27]	2015	Inference	Quantization	L5	Compute	no	CIFAR10, MNIST
[S28]	2018	Inference	Quantization	L4	Compute	yes	ImageNet
[S29]	2018	Inference	Quantization	L3	Energy	no	CIFAR10, MNIST
[S30]	2018	Inference	Quantization	L4	Energy	no	CIFAR10, ImageNet
[S31]	2018	Inference	Quantization	L4	Energy	no	CIFAR10, ImageNet, MNIST, SVHN
[S32]	2016	Inference	Quantization	L4	Energy	yes	CIFAR10, ImageNet, MNIST
[S33]	2020	Inference	Quantization	L4	Compute	no	ImageNet
[S34]	2020	Inference	Quantization	L4	Energy	no	ImageNet
[S35]	2019	Inference	Quantization	L4	Compute	no	CIFAR10, ImageNet
[S36]	2017	Inference	Quantization	L3	Energy	no	CIFAR10, MNIST, SVHN
[S37]	2019	Inference	Quantization	L4	Energy	no	CIFAR10, ImageNet, MNIST
[S38]	2019	Inference	Quantization	L4	Energy	no	ImageNet
[S39]	2019	Inference	BNN	L4	Energy	no	CIFAR10, ImageNet, SVHN
[S40]	2019	Inference	BNN	L4	Compute	no	BENN, CIFAR10, ImageNet
[S41]	2021	Inference	Teacher-Student	L4	Compute	yes	CIFAR10, ImageNet, MNIST
[S42]	2017	Inference	Teacher-Student	L3	Energy	no	CIFAR10, GTSRB, MNIST, SVHN
[S43]	2019	Inference	Teacher-Student	L4	Energy	yes	Caltech-UCSD-Birds, CIFAR10, ImageNet, Scene
[S44]	2015	Inference	Early exit	L3	Energy	no	MNIST
[S45]	2020	Inference	Early exit	L3	Energy	no	CIFAR10, CINIC10
[S46]	2016	Inference	Early exit	L3	Energy	yes	MNIST
[S47]	2020	Inference	Early exit	L4	Energy	yes	CIFAR10, ImageNet, MNIST, SVHN
[S48]	2020	Inference	BNN	L4	Energy	no	CIFAR10, ImageNet, MNIST
[S49]	2016	Inference	Early exit	L3	Compute	no	CIFAR10, MNIST
[S50]	2018	Inference	Compact CNNs	L4	Energy	yes	ImageNet
[S51]	2017	Inference	Compact CNNs	L4	Energy	yes	KITTI
[S52]	2018	Inference	Usage patterns	L4	Compute	yes	Librispeech, Other
[S53]	2020	Inference	Usage patterns	L4	Compute	no	ImageNet
[S54]	2017	Feature selection	Feature selection	L4	Compute	no	Other
[S55]	2021	Feature selection	Feature selection	L3	Compute	no	Other
[S56]	2021	Feature selection	Feature selection	L4	Compute	no	Other
[S57]	2016	Training	Training	L5	Energy	yes	ImageNet
[S58]	2018	Training	Training	L4	Compute	no	ImageNet
[S59]	2020	Training	Training	L3	Energy	no	CIFAR10
[S60]	2017	Training	Training	L3	Compute	no	CONVEX, MNIST, NORB, RECTANGLES
[S61]	2019	Training	Training	L3	Compute	yes	CIFAR10
[S62]	2017	Training	Training	L3	Energy	yes	CIFAR10, MNIST
[S63]	2019	Hyper-parameter	Tuning	L4	Energy	no	ImageNet
[S64]	2018	Hyper-parameter	Tuning	L4	Compute	yes	CIFAR10, Har, ImageNet, MNIST, UbiSound
[S65]	2020	Hyper-parameter	Tuning	L3	Compute	no	CIFAR10
[S66]	2019	Edge offloading	Training	L3	Energy	no	CIFAR10

TABLE 2. *(Continued.)* **Information extracted from papers. The summaries of information are represented as follows: The publication years in Fig.3, and the publication years for edge offloading and hardware acceleration cateogries in Fig.7 and Fig.8, respectively; by category and subcategory in Fig.6; by evidence level and category in Fig.5; and public datasets used in Fig.4. CNN appears in 34% of the papers (cf. Section VI-D) and energy, compute, and labor values of the Optimize column represent 61%, 36%, and 3%, respectively (cf. Section IV).**

[S67]	2020	Edge offloading	Training	L3	Compute	no	MNIST
[S68]	2021	Edge offloading	Training	L3	Energy	no	Other
[S69]	2019	Edge offloading	Training	L4	Compute	no	CIFAR10, ImageNet, MNIST
[S70]	2021	Edge offloading	Training	L3	Energy	no	CIFAR10, MNIST
[S71]	2019	Edge offloading	Training	L4	Compute	no	CIFAR10, ImageNet, MNIST
[S72]	2021	Edge offloading	Training	L3	Compute	no	MNIST
[S73]	2020	Edge offloading	Training	L4	Compute	no	CIFAR10, ImageNet
[S74]	2019	Edge offloading	Inference	L3	Energy	no	CIFAR10, MNIST, Other
[S75]	2018	Edge offloading	Inference	L4	Energy	no	CIFAR10, ImageNet, VOC
[S76]	2017	Edge offloading	Inference	L4	Energy	no	Other
[S77]	2019	Edge offloading	Inference	L4	Energy	no	Other
[S78]	2021	Edge offloading	Inference	L4	Energy	no	ImageNet
[S79]	2017	Edge offloading	Inference	L4	Compute	no	ImageNet
[S80]	2017	Edge offloading	Inference	L4	Energy	no	ImageNet
[S81]	2018	Edge offloading	Inference	L3	Compute	no	CIFAR10
[S82]	2016	Edge offloading	Inference	L4	Energy	yes	Caltech101, CIFAR10, ImageNet, MNIST, STL10
[S83]	2015	Edge offloading	Inference	L4	Energy	no	ImageNet, MNIST
[S84]	2016	Edge offloading	Inference	L3	Energy	yes	CIFAR10, MNIST, SVHN
[S85]	2021	Edge offloading	Both	L4	Compute	no	ImageNet
[S86]	2018	Hardware acceleration	Algorithm co-design	L4	Compute	no	ImageNet
[S87]	2020	Hardware acceleration	Algorithm co-design	L4	Energy	no	CIFAR10, ImageNet, MNIST
[S88]	2017	Hardware acceleration	Algorithm co-design	L4	Compute	yes	CIFAR10, ImageNet, MNIST, SVHN
[S89]	2017	Hardware acceleration	Algorithm co-design	L3	Energy	no	MNIST, SVHN
[S90]	2018	Hardware acceleration	Algorithm co-design	L3	Compute	no	CIFAR10, MNIST
[S91]	2014	Hardware acceleration	Algorithm co-design	L3	Energy	no	CIFAR10, MNIST, SVHN
[S92]	2018	Hardware acceleration	Algorithm co-design	L3	Energy	yes	MNIST
[S93]	2020	Hardware acceleration	Algorithm co-design	L3	Compute	yes	MNIST
[S94]	2017	Hardware acceleration	New numeric formats	L3	Compute	no	
[S95]	2018	Hardware acceleration	New numeric formats	L4	Compute	yes	CIFAR10, ImageNet, MNIST
[S96]	2021	Hardware acceleration	New numeric formats	L4	Energy	no	CIFAR10, ImageNet, MNIST, PTB
[S97]	2017	Hardware acceleration	New numeric formats	L5	Energy	no	CIFAR10, ImageNet, LSUN
[S98]	2019	Hardware acceleration	New numeric formats	L3	Energy	no	CIFAR10, MNIST
[S99]	2018	Hardware acceleration	New numeric formats	L4	Energy	no	CIFAR10, ImageNet
[S100]	2014	Hardware acceleration	Accelerator	L3	Energy	yes	MNIST
[S101]	2017	Hardware acceleration	Accelerator	L4	Energy	yes	ImageNet
[S102]	2019	Hardware acceleration	Accelerator	L4	Energy	no	ImageNet
[S103]	2016	Hardware acceleration	Accelerator	L4	Energy	no	ImageNet
[S104]	2018	Hardware acceleration	Accelerator	L4	Energy	no	ImageNet
[S105]	2017	Hardware acceleration	Accelerator	L4	Energy	no	ImageNet, PTB
[S106]	2016	Hardware acceleration	Accelerator	L4	Energy	yes	CIFAR10, ImageNet, MNIST
[S107]	2018	Hardware acceleration	Accelerator	L4	Compute	yes	ImageNet
[S108]	2017	Hardware acceleration	Accelerator	L4	Energy	yes	ImageNet
[S109]	2017	Hardware acceleration	Algorithm co-design	L4	Compute	yes	ImageNet
[S110]	2014	Human effort	Reduction	L3	Labor	no	MNIST
[S111]	2017	Human effort	Reduction	L4	Labor	yes	CACD, Caltech256
[S112]	2018	Human effort	Reduction	L4	Labor	yes	ImageNet, VOC

It is possible that we missed papers that currently have more citations than at the time of the search. We had to draw the line to an exact date to make the process repeatable. We decided to include papers published until 2021.

We excluded papers on hardware acceleration that did not have commercial implementations at the time of the search. The excluded hardware acceleration approaches may become commercially available in the future, although interest in hardware acceleration has been decreasing (see Fig. 8).

Our search resulted in a sample of published studies, but additional details may have been found with different searches that remain relatively unexplored. For instance, training DNNs with repeated forward and backward propagation steps is a far more memory- and computation-intensive procedure than inference that involves only forward propagation. Considering the cost involved, it is surprising that only 14% of the primary studies are about training (in the cloud or at the edge), and the majority of the primary studies are about inference (in the cloud or at the edge or accelerated by hardware). However, we also aimed to point out these topics as relevant directions for future work to be considered with respect to cost.

Our categorization of papers is mutually exclusive to make it easier to follow, although some papers can be assigned to multiple categories, such as pruning and edge inference. This makes the exact number of papers in each category less interesting. Even with some movements between categories,

the relative sizes of the categories remain approximately the same.

Only four primary studies are from the industry (see evidence levels 5 and 6 in Fig. 5). It is unclear how widely the methods reviewed in this paper are used in practice, although the availability of the methods in popular DL libraries provides a hint. Finally, paper selection was conducted primarily by the first author, which probably introduced some bias in the included and excluded papers.

VIII. CONCLUSION

We presented a systematic literature review on methods for making deep learning more cost-efficient in monetary terms for software development organizations. We selected and analyzed 112 primary studies resulting in a two-level categorization of the methods for cost-efficient DL. We found that inference optimization is a more popular topic in the literature than training optimization. Hardware acceleration is heavily represented because the general-purpose Von Neumann processor architecture, owing to its memory bottleneck, is far from optimal for DNN execution. We discovered that image inputs dominate the research on deep learning optimizations, and we claim that there is a research gap related to input types other than images for DNNs. Likewise, the research literature is dominated by open datasets with only a few industrial proof-of-concept or practical studies. Processing parts of DNNs in client devices in the network edge is currently under active research and has significant potential for cost savings and deserves further research.

APPENDIX A
INFORMATION EXTRACTED FROM PAPERS

See Table 2.

PRIMARY STUDIES

[S1] H. Alfauri and F. Esposito, "A distributed consensus protocol for sustainable federated learning," in *2021 IEEE 7th International Conference on Network Softwarization (NetSoft)*, Jun. 2021, pp. 161–165.

[S2] G. Guo and J. Zhang, "Energy-efficient incremental offloading of neural network computations in mobile edge computing," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, Dec. 2020, pp. 1–6.

[S3] J. H. Ko, T. Na, M. F. Amir, and S. Mukhopadhyay, "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained Internet-of-Things platforms," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, Nov. 2018, pp. 1–6.

[S4] N. H. Tran, W. Bao, A. Zomaya, M. N. H. Nguyen, and C. S. Hong, "Federated learning over wireless networks: Optimization model design and analysis," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, Apr. 2019, pp. 1387–1395.

[S5] X. Wang, M. Magno, L. Cavigelli, and L. Benini, "FANN-on-MCU: An open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4403–4417, May 2020.

[S6] Z. Xu, L. Zhao, W. Liang, O. F. Rana, P. Zhou, Q. Xia, W. Xu, and G. Wu, "Energy-aware inference offloading for DNN-driven applications in mobile edge clouds," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 4, pp. 799–814, Apr. 2021.

[S7] Z. Du, R. Fasthuber, T. Chen, P. Ienne, L. Li, T. Luo, X. Feng, Y. Chen, and O. Temam, "ShiDianNao: Shifting vision processing closer to the sensor," in *2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2015, pp. 92–104.

[S8] A. E. Eshratifar and M. Pedram, "Energy and performance efficient computation offloading for deep neural networks in a mobile cloud computing environment," in *Proceedings of the 2018 on Great Lakes Symposium on VLSI*, ser. GLSVLSI '18. New York, NY, USA: Association for Computing Machinery, May 2018, pp. 111–116.

[S9] N. K. Jha and S. Mittal, "Modeling data reuse in deep neural networks by taking data-types into cognizance," *IEEE Transactions on Computers*, vol. 70, no. 9, pp. 1526–1538, Sep. 2021.

[S10] B. Lin, Y. Huang, J. Zhang, J. Hu, X. Chen, and J. Li, "Cost-driven offloading for DNN-based applications over cloud, edge, and end devices," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 8, pp. 5456–5466, Aug. 2020.

[S11] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '15. New York, NY, USA: Association for Computing Machinery, Feb. 2015, pp. 161–170.

[S12] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. L. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a Tensor Processing Unit," in *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ser. ISCA '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–12.

[S13] S. S. Sarwar, G. Srinivasan, B. Han, P. Wijesinghe, A. Jaiswal, P. Panda, A. Raghunathan, and K. Roy, "Energy efficient neural computing: A study of cross-layer approximations," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 4, pp. 796–809, Dec. 2018.

[S14] X. Ma, S. Lin, S. Ye, Z. He, L. Zhang, G. Yuan, S. Tan, Z. Li, D. Fan, X. Qian, X. Lin, K. Ma, and Y. Wang, "Non-structured DNN weight pruning—Is it beneficial in any platform?" *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 9, pp. 4930–4944, Sep. 2022.

[S15] Q. Qin, J. Ren, J. Yu, H. Wang, L. Gao, J. Zheng, Y. Feng, J. Fang, and Z. Wang, "To compress, or not to compress: Characterizing deep learning model compression for embedded inference," in *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*, Dec. 2018, pp. 729–736.

[S16] T. He, Y. Fan, Y. Qian, T. Tan, and K. Yu, "Reshaping deep neural network for fast decoding by node-pruning," in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2014, pp. 245–249.

[S17] Z. Wang, C. Zhu, Z. Xia, Q. Guo, and Y. Liu, "Towards thinner convolutional neural networks through gradually global pruning," in *2017 IEEE International Conference on Image Processing (ICIP)*, Sep. 2017, pp. 3939–3943.

[S18] M. Riera, J. Arnau, and A. Gonzez, "DNN pruning with principal component analysis and connection importance estimation," *Journal of Systems Architecture*, vol. 122, 2022.

[S19] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 6071–6079.

[S20] X. Zhou, Z. Du, Q. Guo, S. Liu, C. Liu, C. Wang, X. Zhou, L. Li, T. Chen, and Y. Chen, "Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach," in *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2018, pp. 15–28.

[S21] X. Xu, Q. Chen, L. Xie, and H. Su, "Batch-normalization-based soft filter pruning for deep convolutional neural networks," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, Dec. 2020, pp. 951–956.

[S22] J. Zhang, X. Chen, M. Song, and T. Li, "Eager pruning: Algorithm and architecture support for fast training of deep neural networks," in *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2019, pp. 292–303.

[S23] X. Dai, H. Yin, and N. Jha, "Incremental learning using a grow-and-prune paradigm with efficient neural networks," *IEEE Transactions on Emerging Topics in Computing*, vol. 10, no. 2, pp. 752–762, Apr. 2022.

[S24] B. O. Ayinde, T. Inanc, and J. M. Zurada, "Redundant feature pruning for accelerated inference in deep neural networks," *Neural Networks*, vol. 118, pp. 148–158, Oct. 2019.

[S25] J. Guo and M. Potkonjak, "Pruning filters and classes: Towards on-device customization of convolutional neural networks," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*, ser. EMDL '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 13–17.

[S26] R. Yazdani, M. Riera, J. M. Arnau, and A. Gonzlez, "The dark side of DNN pruning," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 790–801.

[S27] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning*, vol. 3. PMLR, Jun. 2015, pp. 1737–1746. [Online]. Available: https://proceedings.mlr.press/v37/gupta15.html

[S28] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for MobileNets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Mar. 2018, pp. 14–18.

[S29] R. Ding, Z. Liu, R. D. S. Blanton, and D. Marculescu, "Quantized deep neural networks for energy efficient hardware-based inference," in *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2018, pp. 1–8.

[S30] N. Wang, J. Choi, D. Brand, C. Y. Chen, and K. Gopalakrishnan, "Training deep neural networks with 8-bit floating point numbers," vol. 31, 2018, pp. 7675–7684.

[S31] S. Wu, G. Li, F. Chen, and L. Shi, "Training and inference with integers in deep neural networks," in *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, 2018.

[S32] B. Moons, B. De Brabandere, L. Van Gool, and M. Verhelst, "Energy-efficient ConvNets through approximate computing," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–8.

[S33] S. M. Nabavinejad, L. Mashayekhy, and S. Reda, "ApproxDNN: Incentivizing DNN approximation in cloud," in *2020 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGRID)*, May 2020, pp. 639–648.

[S34] Y. Yang, L. Deng, S. Wu, T. Yan, Y. Xie, and G. Li, "Training high-performance and large-scale deep neural networks with full 8-bit integers," *Neural Networks*, vol. 125, pp. 70–82, May 2020.

[S35] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, Dec. 2019, pp. 52–56.

[S36] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst, "Minimum energy quantized neural networks," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, Oct. 2017, pp. 1921–1925.

[S37] T. Zhang, L. Zhu, Q. Zhao, and K. Shin, "Neural networks weights quantization: Target none-retraining ternary (TNT)," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, Dec. 2019, pp. 62–65.

[S38] S. Sasaki, A. Maki, D. Miyashita, and J. Deguchi, "Post training weight compression with distribution-based filter-wise quantization step," in *2019 IEEE Symposium in Low-Power and High-Speed Chips (COOL CHIPS)*, Apr. 2019, pp. 1–3.

[S39] R. Ding, T. W. Chin, Z. Liu, and D. Marculescu, "Regularizing activation distribution for training binarized deep networks," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 11 400–11 409.

[S40] S. Zhu, X. Dong, and H. Su, "Binary ensemble neural network: More bits per network or more networks per bit?" in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, vol. 2019-June, Jun. 2019, pp. 4918–4927.

[S41] Z. Tao, Q. Xia, and Q. Li, "Neuron manifold distillation for edge deep learning," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, Jun. 2021, pp. 1–10.

[S42] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Petrot, "Ternary neural networks for resource-efficient AI applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 2547–2554.

[S43] K. Bhardwaj, C. Y. Lin, A. Sartor, and R. Marculescu, "Memory- and communication-aware model compression for distributed deep learning inference on IoT," *ACM Transactions on Embedded Computing Systems*, vol. 18, no. 5, pp. 82:1–82:22, Oct. 2019.

[S44] S. Venkataramani, A. Raghunathan, J. Liu, and M. Shoaib, "Scalable-effort classifiers for energy-efficient machine learning," in *Proceedings of the 52nd Annual Design Automation Conference*, ser. DAC '15. New York, NY, USA: Association for Computing Machinery, Jun. 2015, pp. 1–6.

[S45] S. Scardapane, D. Comminiello, M. Scarpiniti, E. Baccarelli, and A. Uncini, "Differentiable branching in deep networks for fast inference," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2020, pp. 4167–4171.

[S46] P. Panda, A. Sengupta, and K. Roy, "Conditional deep learning for energy-efficient and enhanced pattern recognition," in *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Mar. 2016, pp. 475–480.

[S47] N. K. Jayakodi, S. Belakaria, A. D., and J. R. Doppa, "Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models," *ACM Transactions on Embedded Computing Systems*, vol. 19, no. 1, pp. 4:1–4:24, Feb. 2020.

[S48] Y. Huang, X. Qiao, P. Ren, L. Liu, C. Pu, S. Dustdar, and J. Chen, "A lightweight collaborative deep neural network for the mobile web in edge cloud," *IEEE Transactions on Mobile Computing*, vol. 21, no. 7, pp. 2289–2305, Jul. 2022.

[S49] S. Teerapittayanon, B. McDanel, and H. T. Kung, "BranchyNet: Fast inference via early exiting from deep neural networks," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, Dec. 2016, pp. 2464–2469.

[S50] A. Gholami, K. Kwon, B. Wu, Z. Tai, X. Yue, P. Jin, S. Zhao, and K. Keutzer, "SqueezeNext: Hardware-aware neural network design," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jun. 2018, pp. 1719–171 909.

[S51] B. Wu, A. Wan, F. Iandola, P. H. Jin, and K. Keutzer, "Squeezedet: Unified, small, low power, fully convolutional neural networks for real-time object detection for autonomous driving," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, Jul. 2017, pp. 446–454.

[S52] M. Riera, J. M. Arnau, and A. Gonzalez, "Computation reuse in DNNs by exploiting input similarity," in *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2018, pp. 57–68.

[S53] A. Ali, R. Pinciroli, F. Yan, and E. Smirni, "BATCH: Machine learning inference serving on serverless platforms with adaptive batching," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2020, pp. 1–15.

[S54] J. Xu, B. Tang, H. He, and H. Man, "Semisupervised feature selection based on relevance and redundancy criteria," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 9, pp. 1974–1984, Sep. 2017.

[S55] Y. Chen, Y. Wang, L. Cao, and Q. Jin, "CCFS: A confidence-based cost-effective feature selection scheme for healthcare data classification," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 3, pp. 902–911, May 2021.

[S56] L. Jiang, G. Kong, and C. Li, "Wrapper framework for test-cost-sensitive feature selection," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 51, no. 3, pp. 1747–1756, Mar. 2021.

[S57] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler, "vDNN: Virtualized deep neural networks for scalable, memory-efficient neural network design," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-49. Taipei, Taiwan: IEEE Press, Oct. 2016, pp. 1–13.

[S58] L. Wang, J. Ye, Y. Zhao, W. Wu, A. Li, S. L. Song, Z. Xu, and T. Kraska, "Superneurons: Dynamic GPU memory management for training deep neural networks," in *Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, ser. PPoPP '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 41–53.

[S59] X. Zhang, J. Wang, G. Joshi, and C. Joe-Wong, "Machine learning on volatile instances," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, Jul. 2020, pp. 139–148.

[S60] R. Spring and A. Shrivastava, "Scalable and sustainable deep learning via randomized hashing," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '17. New York, NY, USA: Association for Computing Machinery, Aug. 2017, pp. 445–454.

[S61] Z. Wang, S. H. Nelaturu, and S. Amarasinghe, "Accelerated CNN training through gradient approximation," in *2019 2nd Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Feb. 2019, pp. 31–35.

[S62] S. S. Sarwar, P. Panda, and K. Roy, "Gabor filter assisted energy efficient fast learning convolutional neural networks," in *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Jul. 2017, pp. 1–6.

[S63] X. Dai, P. Zhang, B. Wu, H. Yin, F. Sun, Y. Wang, M. Dukhan, Y. Hu, Y. Wu, Y. Jia, P. Vajda, M. Uyttendaele, and N. K. Jha, "ChamNet: Towards efficient network design through platform-aware model adaptation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 11 390–11 399.

[S64] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 389–400.

[S65] Y. Li, B. An, J. Ma, D. Cao, Y. Wang, and H. Mei, "SpotTune: Leveraging transient resources for cost-efficient hyper-parameter tuning in the public cloud," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, Nov. 2020, pp. 45–55.

[S66] B. Gu, J. Kong, A. Munir, and Y. G. Kim, "A framework for distributed deep neural network training with heterogeneous computing platforms," in *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2019, pp. 430–437.

[S67] S. Luo, X. Chen, Q. Wu, Z. Zhou, and S. Yu, "HFEL: Joint edge association and resource allocation for cost-efficient hierarchical federated edge learning," *IEEE Transactions on Wireless Communications*, vol. 19, no. 10, pp. 6535–6548, Oct. 2020.

[S68] Z. Yang, M. Chen, W. Saad, C. S. Hong, and M. Shikh-Bahaei, "Energy efficient federated learning over wireless communication networks," *IEEE Transactions on Wireless Communications*, vol. 20, no. 3, pp. 1935–1949, Mar. 2021.

[S69] L. Song, J. Mao, Y. Zhuo, X. Qian, H. Li, and Y. Chen, "HyPar: Towards hybrid parallelism for deep learning accelerator array," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2019, pp. 56–68.

[S70] L. Li, D. Shi, R. Hou, H. Li, M. Pan, and Z. Han, "To talk or to work: Flexible communication compression for energy efficient federated learning over heterogeneous mobile edge devices," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 2021, pp. 1–10.

[S71] F. Sattler, S. Wiedemann, K. R. Mller, and W. Samek, "Sparse binary compression: Towards distributed deep learning with minimal communication," in *2019 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2019, pp. 1–8.

[S72] B. Luo, X. Li, S. Wang, J. Huang, and L. Tassiulas, "Cost-effective federated learning design," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 2021, pp. 1–10.

[S73] Q. Luo, J. He, Y. Zhuo, and X. Qian, "Prague: High-performance heterogeneity-aware asynchronous decentralized training," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, Mar. 2020, pp. 401–416.

[S74] S. Gopinath, N. Ghanathe, V. Seshadri, and R. Sharma, "Compiling KB-sized machine learning models to tiny IoT devices," in *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, ser. PLDI 2019. New York, NY, USA: Association for Computing Machinery, Jun. 2019, pp. 79–95.

[S75] D. Kang, E. Kim, I. Bae, B. Egger, and S. Ha, "C-GOOD: C-code generation framework for optimized on-device deep learning," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18. New York, NY, USA: Association for Computing Machinery, Nov. 2018, pp. 1–8.

[S76] S. Teerapittayanon, B. McDanel, and H. Kung, "Distributed deep neural networks over the cloud, the edge and end devices," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, 2017, pp. 328–339.

[S77] A. Thomas, Y. Guo, Y. Kim, B. Aksanli, A. Kumar, and T. S. Rosing, "Hierarchical and distributed machine learning inference beyond the edge," in *2019 IEEE 16th International Conference on Networking, Sensing and Control (ICNSC)*, May 2019, pp. 18–23.

[S78] L. Zeng, X. Chen, Z. Zhou, L. Yang, and J. Zhang, "CoEdge: Cooperative DNN inference with adaptive workload partitioning over heterogeneous edge devices," *IEEE/ACM Transactions on Networking*, vol. 29, no. 2, pp. 595–608, Apr. 2021.

[S79] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "RSTensorFlow: GPU enabled TensorFlow for deep learning on commodity Android devices," in *Proceedings of the 1st International Workshop on Deep Learning for Mobile Systems and Applications*, ser. EMDL '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 7–12.

[S80] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGPLAN Notices*, vol. 52, no. 4, pp. 615–629, 2017.

[S81] E. Li, Z. Zhou, and X. Chen, "Edge intelligence: On-demand deep learning model co-inference with device-edge synergy," in *Proceedings of the 2018 Workshop on Mobile Edge Communications*, ser. MECOMM'18. New York, NY, USA: Association for Computing Machinery, Aug. 2018, pp. 31–36.

[S82] G. Hegde, Siddhartha, N. Ramasamy, and N. Kapre, "CaffePresso: An optimized library for deep learning on embedded accelerator-based platforms," in *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, Oct. 2016, pp. 1–10.

[S83] E. Park, D. Kim, S. Kim, Y. D. Kim, G. Kim, S. Yoon, and S. Yoo, "Big/little deep neural network for ultra low power inference," in *2015 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, Oct. 2015, pp. 124–132.

[S84] S. Tann, H.and Hashemi, R. I. Bahar, and S. Reda, "Runtime configurable deep neural networks for energy-accuracy trade-off," in *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software codesign and System Synthesis*, ser. CODES '16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1–10.

[S85] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 565–576, Feb. 2021.

[S86] D. Lee, S. Kang, and K. Choi, "ComPEND: Computation pruning through early negative detection for ReLU in a deep neural network accelerator," in *Proceedings of the 2018 International Conference on Supercomputing*, ser. ICS '18. New York, NY, USA: Association for Computing Machinery, Jun. 2018, pp. 139–148.

[S87] Y. Zhao, X. Chen, Y. Wang, C. Li, H. You, Y. Fu, Y. Xie, Z. Wang, and Y. Lin, "SmartExchange: Trading higher-cost memory storage/access for lower-cost computation," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, May 2020, pp. 954–967.

[S88] C. Ding, S. Liao, Y. Wang, Z. Li, N. Liu, Y. Zhuo, C. Wang, X. Qian, Y. Bai, G. Yuan, X. Ma, Y. Zhang, J. Tang, Q. Qiu, X. Lin, and B. Yuan, "CIRCNN: Accelerating and compressing deep neural networks using block-circulant weight matrices," in *Proceedings of the Annual International Symposium on Microarchitecture, MICRO*, vol. Part F131207, 2017, pp. 395–408.

[S89] S. Liao, Z. Li, X. Lin, Q. Qiu, Y. Wang, and B. Yuan, "Energy-efficient, high-performance, highly-compressed deep neural network design using block-circulant matrices," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2017, pp. 458–465.

[S90] S. Lin, N. Liu, M. Nazemi, H. Li, C. Ding, Y. Wang, and M. Pedram, "FFT-based deep learning deployment in embedded systems," in *2018 Design, Automation Test in Europe Conference exhibition (DATE)*, Mar. 2018, pp. 1045–1050.

[S91] S. Venkataramani, A. Ranjan, K. Roy, and A. Raghunathan, "AxNN: Energy-efficient neuromorphic systems using approximate computing," in *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Aug. 2014, pp. 27–32.

[S92] X. Jiao, V. Akhlaghi, Y. Jiang, and R. K. Gupta, "Energy-efficient neural networks using approximate computation reuse," in *2018 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2018, pp. 1223–1228.

[S93] E. Dupuis, D. Novo, I. O'Connor, and A. Bosio, "On the automatic exploration of weight sharing for deep neural network compression," in *2020 Design, Automation Test in Europe Conference Exhibition (DATE)*, Mar. 2020, pp. 1319–1322.

[S94] J. L. Gustafson and I. Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, pp. 71–86, 2017.

[S95] S. H. Fatemi Langroudi, T. Pandit, and D. Kudithipudi, "Deep learning inference on embedded devices: Fixed-point vs Posit," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*, Mar. 2018, pp. 19–23.

[S96] J. Lu, C. Fang, M. Xu, J. Lin, and Z. Wang, "Evaluations on deep neural networks training using Posit number system," *IEEE Transactions on Computers*, vol. 70, no. 2, pp. 174–187, Feb. 2021.

[S97] U. ster, T. J. Webb, X. Wang, M. Nassar, A. K. Bansal, W. H. Constable, O. H. Elibol, S. Gray, S. Hall, L. Hornof, A. Khosrowshahi, C. Kloss, R. J. Pai, and N. Rao, "Flexpoint: An adaptive numerical format for efficient training of deep neural networks," Dec. 2017. [Online]. Available: http://arxiv.org/abs/1711.02213

[S98] P. M. Tostado, B. U. Pedroni, and G. Cauwenberghs, "Performance trade-offs in weight quantization for memory-efficient inference," in *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*, Mar. 2019, pp. 246–250.

[S99] S. Jain, S. Venkataramani, V. Srinivasan, J. Choi, P. Chuang, and L. Chang, "Compensated-DNN: Energy efficient low-precision deep neural networks by compensating quantization errors," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, Jun. 2018, pp. 1–6.

[S100] T. Chen, Z. Du, N. Sun, J. Wang, C. Wu, Y. Chen, and O. Temam, "DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning," in *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, ser. ASPLOS '14. New York, NY, USA: Association for Computing Machinery, Feb. 2014, pp. 269–284.

[S101] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[S102] Y.-H. Chen, T.-J. Yang, J. Emer, and V. Sze, "Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 2, pp. 292–308, Jun. 2019.

[S103] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2016, pp. 243–254.

[S104] H. Kwon, A. Samajdar, and T. Krishna, "MAERI: Enabling flexible dataflow mapping over DNN accelerators via reconfigurable interconnects," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*. New York, NY, USA: Association for Computing Machinery, Mar. 2018, pp. 461–475.

[S105] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, "FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates," in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2017, pp. 152–159.

[S106] H. Sharma, J. Park, D. Mahajan, E. Amaro, J. K. Kim, C. Shao, A. Mishra, and H. Esmaeilzadeh, "From high-level deep neural models to FPGAs," in *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Oct. 2016, pp. 1–12.

[S107] X. Zhang, J. Wang, C. Zhu, Y. Lin, J. Xiong, W. M. Hwu, and D. Chen, "DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs," in *Proceedings of the International Conference on Computer-Aided Design*, ser. ICCAD '18, New York, NY, USA, Nov. 2018, pp. 1–8.

[S108] Y. Ma, Y. Cao, S. Vrudhula, and J. S. Seo, "Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, ser. FPGA '17. New York, NY, USA: Association for Computing Machinery, Feb. 2017, pp. 45–54.

[S109] X. Wei, C. H. Yu, P. Zhang, Y. Chen, Y. Wang, H. Hu, Y. Liang, and J. Cong, "Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs," in *Proceedings of the 54th Annual Design Automation Conference 2017*, ser. DAC '17. New York, NY, USA: Association for Computing Machinery, Jun. 2017, pp. 1–6.

[S110] D. Wang and Y. Shang, "A new active labeling method for deep learning," in *2014 International Joint Conference on Neural Networks (IJCNN)*, Jul. 2014, pp. 112–119.

[S111] K. Wang, D. Zhang, Y. Li, R. Zhang, and L. Lin, "Cost-effective active learning for deep image classification," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 27, no. 12, pp. 2591–2600, Dec. 2017.

[S112] S. J. Huang, J. W. Zhao, and Z. Y. Liu, "Cost-effective training of deep CNNs with active model adaptation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '18. New York, NY, USA: Association for Computing Machinery, Jul. 2018, pp. 1580–1588. [Online]. Available: https://doi.org/10.1145/3219819.3220026

REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[2] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Workshop Mobile Cloud Comput.*, New York, NY, USA, Aug. 2012, pp. 13–16.

[4] C. J. Wu et al., "Machine learning at Facebook: Understanding inference at the edge," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2019, pp. 331–344.

[5] K. Hazelwood, S. Bird, D. Brooks, S. Chintala, U. Diril, D. Dzhulgakov, M. Fawzy, B. Jia, Y. Jia, A. Kalro, J. Law, K. Lee, J. Lu, P. Noordhuis, M. Smelyanskiy, L. Xiong, and X. Wang, "Applied machine learning at Facebook: A datacenter infrastructure perspective," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 620–629.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, vol. 25. Red Hook, NY, USA: Curran Associates, 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

[7] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *IEEE Int. Solid-State Circuits Conf. (ISSCC) Dig. Tech. Papers*, Feb. 2014, pp. 10–14.

[8] W. A. Wulf and S. A. McKee, "Hitting the memory wall: Implications of the obvious," *ACM SIGARCH Comput. Archit. News*, vol. 23, no. 1, pp. 20–24, Mar. 1995.

[9] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, and S. Bates, "In-datacenter performance analysis of a tensor processing unit," in *Proc. ACM/IEEE 44th Annu. Int. Symp. Comput. Archit. (ISCA)*, Jun. 2017, pp. 1–12.

[10] J. Hestness, S. Narang, N. Ardalani, G. Diamos, H. Jun, H. Kianinejad, M. M. A. Patwary, Y. Yang, and Y. Zhou, "Deep learning scaling is predictable, empirically," 2017, *arXiv:1712.00409*.

[11] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 12, pp. 2346–2363, Dec. 2019.

[12] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. Ranzato, A. Senior, P. Tucker, K. Yang, Q. Le, and A. Ng, "Large scale distributed deep networks," in *Advances in Neural Information Processing Systems*, vol. 25. Red Hook, NY, USA: Curran Associates, 2012.

[13] H. Zheng, F. Xu, L. Chen, Z. Zhou, and F. Liu, "*Cynthia*: Cost-efficient cloud resource provisioning for predictable distributed deep neural network training," in *Proc. 48th Int. Conf. Parallel Process.*, Aug. 2019, pp. 1–11.

[14] B. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Softw. Eng. Group, School Comput. Sci. Math., Keele Univ., Dept. Comput. Sci., Univ. Durham, EBSE Tech. Rep., Version 2, 2007.

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[16] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27. Red Hook, NY, USA: Curran Associates, 2014.

[17] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2017, *arXiv:1609.02907*.

[18] R. Schwartz, J. Dodge, N. Smith, and O. Etzioni, "Green AI," *Commun. ACM*, vol. 63, no. 12, pp. 54–63, Dec. 2020.

[19] E. García-Martín, C. F. Rodrigues, G. Riley, and H. Grahn, "Estimation of energy consumption in machine learning," *J. Parallel Distrib. Comput.*, vol. 134, pp. 75–88, Dec. 2019.

[20] D. He, Z. Wang, and J. Liu, "A survey to predict the trend of AI-able server evolution in the cloud," *IEEE Access*, vol. 6, pp. 10591–10602, 2018.

[21] A. Shawahna, S. M. Sait, and A. El-Maleh, "FPGA-based accelerators of deep learning networks for learning and classification: A review," *IEEE Access*, vol. 7, pp. 7823–7859, 2019.

[22] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proc. IEEE*, vol. 105, no. 12, pp. 2295–2329, Dec. 2017.

[23] V. Sze, Y.-H. Chen, J. Emer, A. Suleiman, and Z. Zhang, "Hardware for machine learning: Challenges and opportunities," in *Proc. IEEE Custom Integr. Circuits Conf. (CICC)*, Apr. 2018, pp. 1–8.

[24] M. Verhelst and B. Moons, "Embedded deep neural network processing: Algorithmic and processor techniques bring deep learning to IoT and edge devices," *IEEE Solid StateCircuits Mag.*, vol. 9, no. 4, pp. 55–65, Fall 2017.

[25] M. Shafique, T. Theocharides, C.-S. Bouganis, M. A. Hanif, F. Khalid, R. Hafiz, and S. Rehman, "An overview of next-generation architectures for machine learning: Roadmap, opportunities and challenges in the IoT era," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 827–832.

[26] A. Ibrahim, M. Osta, M. Alameh, M. Saleh, H. Chible, and M. Valle, "Approximate computing methods for embedded machine learning," in *Proc. 25th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2018, pp. 845–848.

[27] M. Capra, B. Bussolino, A. Marchisio, G. Masera, M. Martina, and M. Shafique, "Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead," *IEEE Access*, vol. 8, pp. 225134–225180, 2020.

[28] L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, "Model compression and hardware acceleration for neural networks: A comprehensive survey," *Proc. IEEE*, vol. 108, no. 4, pp. 485–532, Apr. 2020.

[29] T. Liang, J. Glossner, L. Wang, S. Shi, and X. Zhang, "Pruning and quantization for deep neural network acceleration: A survey," *Neurocomputing*, vol. 461, pp. 370–403, Oct. 2021.

[30] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision: Improving the Efficiency of Artificial Intelligence*, Jun. 2021. [Online]. Available: https://arxiv.org/abs/2103.13630

[31] A. Marchisio, M. A. Hanif, F. Khalid, G. Plastiras, C. Kyrkou, T. Theocharides, and M. Shafique, "Deep learning for edge computing: Current trends, cross-layer optimizations, and open research challenges," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 553–559.

[32] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[33] (2022). *IEEE Xplore*. [Online]. Available: https://developer.ieee.org/

[34] V. Alves, N. Niu, C. Alves, and G. Valença, "Requirements engineering for software product lines: A systematic literature review," *Inf. Softw. Technol.*, vol. 52, no. 8, pp. 806–820, Aug. 2010.

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

[36] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient ConvNets," 2016, *arXiv:1608.08710*.

[37] M. Courbariaux, Y. Bengio, and J. P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems*, vol. 28. Red Hook, NY, USA: Curran Associates, 2015. [Online]. Available: https://proceedings.neurips.cc/paper/2015/hash/3e15cc11f979ed25912dff5b0669f2cd-Abstract.html

[38] W. Sung, S. Shin, and K. Hwang, "Resiliency of deep neural networks under quantization," 2015, *arXiv:1511.06488*.

[39] M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 1–12.

[40] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in Neural Information Processing Systems*, vol. 27. Red Hook, NY, USA: Curran Associates, 2014. [Online]. Available: https://proceedings.neurips.cc/paper/2014/hash/ea8fcd92d59581717e06eb187f10666d-Abstract.html

[41] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *Proc. NIPS Deep Learn. Workshop*, 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[42] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv:1704.04861*.

[43] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, Jan. 2003.

[44] TensorFlow Lite. (2022). *ML for Mobile and Edge Devices*. [Online]. Available: https://www.tensorflow.org/lite

[45] R. David, J. Duke, A. Jain, V. J. Reddi, N. Jeffries, J. Li, N. Kreeger, I. Nappier, M. Natraj, T. Wang, and P. Warden, "Tensorflow lite micro: Embedded machine learning for TinyML systems," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 800–811, Mar. 2021. [Online]. Available: https://proceedings.mlsys.org/paper/2021/hash/d2ddea18f00665ce8623e36bd4e3c7c5-Abstract.html

[46] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, Jan. 2019.

[47] G. Burr, R. Shelby, A. Sebastian, S. Kim, S. Kim, S. Sidler, K. Virwani, M. Ishii, P. Narayanan, A. Fumarola, L. L. Sanches, I. Boybat, M. Le Gallo, K. Moon, J. Woo, H. Hwang, and Y. Leblebici, "Neuromorphic computing using non-volatile memory," *Adv. Phys. X*, vol. 2, no. 1, pp. 89–124, 2017.

[48] Apple. (2022). *Apple Core ML*. [Online]. Available: https://developer.apple.com/machine-learning/core-ml/

[49] Android. (2022). *Android ML*. [Online]. Available: https://developer.android.com/ml

[50] (2022). *TensorFlow*. [Online]. Available: https://www.tensorflow.org/

[51] (2022). *PyTorch*. [Online]. Available: https://www.pytorch.org

[52] L. Corneo, M. Eder, N. Mohan, A. Zavodovski, S. Bayhan, W. Wong, P. Gunningberg, J. Kangasharju, and J. Ott, "Surrounded by the clouds: A comprehensive cloud reachability study," in *Proc. Web Conf.* New York, NY, USA, Apr. 2021, pp. 295–304.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*.
[54] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
[55] S. Ruder, "An overview of gradient descent optimization algorithms," 2016, *arXiv:1609.04747*.

ANTTI KLEMETTI received the M.S. degree in computer science from the University of Helsinki, Finland, in 2001.

He is currently a Ph.D. Researcher in computer science with the University of Helsinki. He is a seasoned Software Practitioner with over 20 years of experience in international companies, such as Nokia and Unity Technologies. He is also working on his industry-inspired Ph.D. thesis on cost-efficient machine learning.

MIKKO RAATIKAINEN received the D.Sc. (Tech.) degree in computer science and engineering from Aalto University, in 2019. He is currently a Researcher with the Empirical Software Engineering Research Group, University of Helsinki. His research interests include empirical research in software engineering and business.

LALLI MYLLYAHO received the M.S. degree in mathematics from the University of Helsinki, Finland, in 2018.

He is currently a Ph.D. Researcher in computer science with the University of Helsinki. He is also a member of the Empirical Software Engineering Group, University of Helsinki, with a background in mathematics and teaching. His current research interests include the reliability and operation of machine learning systems.

TOMMI MIKKONEN received the M.S. and Ph.D. degrees in computer science from the Tampere University of Technology, Finland, in 1992 and 1999, respectively.

He is currently a Full Professor of software engineering with the University of Jyväskylä, Finland. His current research interests include the IoT, software engineering, and multi-device programming.

JUKKA K. NURMINEN (Member, IEEE) received the M.S. and Ph.D. degrees in applied mathematics from the Helsinki University of Technology (now Aalto University), Finland, in 1986 and 2003, respectively.

He has worked extensively on software research in the telecom industry with the Nokia Research Center, in academia with Aalto University, and applied research with VTT. He is currently a Professor of computer science with the University of Helsinki, Finland. His key research contributions are energy-efficient software, mobile peer-to-peer networking, and cloud solutions; however, his experience ranges widely from applied optimization to AI, network planning tools to mobile apps, and software project management to tens of patented inventions. His main research interests include engineering machine learning systems, fair and reliable operation of AI, and software development for quantum computers.