Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Survey Track

A Survey on Efficient Training of Transformers

Bohan Zhuang[1†], **Jing Liu**[1], **Zizheng Pan**[1], **Haoyu He**[1], **Yuetian Weng**[1], **Chunhua Shen**[2]
[1]ZIP Lab, Monash University
[2]Zhejiang University
{bohan.zhuang, jing.liu1, zizheng.pan, haoyu.he, yuetian.weng}@monash.edu, chunhuashen@zju.edu.cn

Abstract

Recent advances in Transformers have come with a huge requirement on computing resources, highlighting the importance of developing efficient training techniques to make Transformer training faster, at lower cost, and to higher accuracy by the efficient use of computation and memory resources. This survey provides the first systematic overview of the efficient training of Transformers, covering the recent progress in acceleration arithmetic and hardware, with a focus on the former. We analyze and compare methods that save computation and memory costs for intermediate tensors during training, together with techniques on hardware/algorithm co-design. We finally discuss challenges and promising areas for future research.

1 Introduction

Deep learning is a recent most profound approach which has revolutionised machine learning (ML) and artificial intelligence and is leading the fourth industrial revolution. At its core, the success of deep learning depends on the vast computational resources available and an extremely large amounts of labeled data. Despite the huge excitement generated by the recent developments, deep learning models, especially Transformers [Vaswani *et al.*, 2017], have become formidably large and computationally intensive, resulting in two pressing challenges at the fundamental level.

The first issue concerns the intensive computation of training large Transformer-based models. A widely discussed energy study of deep learning models [Strubell *et al.*, 2019] estimates that training a Transformer base model with neural architecture search (NAS) [So *et al.*, 2019] produces about 626,155 pounds of planet-warming carbon dioxide, equal to the lifetime emissions of five cars; as models grow bigger, their demand for computing is outpacing improvements in hardware efficiency. For example, GPT-3 [Brown *et al.*, 2020] (the precursor to ChatGPT) was trained on half a trillion words and equips with 175 billion parameters. Notably, according to the technical overview of GPT-3[1], it would take 355 GPU-years and cost at least $4.6M for a single training run, estimated with theoretical 28 TFLOPS for V100 and lowest 3-year reserved cloud pricing. Therefore, the true groundbreaking success of deep learning, such as ChatGPT, is exclusively dominated by large and rich enterprises such as Google or Microsoft. It becomes extremely important to make deep learning tenable in computation and energy efficiency for Green AI [Schwartz *et al.*, 2020], and democratize AI to wider communities with limited resources.

The second issue comes with the exponentially growing training memory proportional to the attention-based model size. For example, the largest language model in literature grows from 345M with BERT-large [Kenton and Toutanova, 2019] in 2018, to hundreds of billions till now with models such as MT-NLG [Smith *et al.*, 2022] equipped with 530B parameters. Therefore, these SOTA massive models call for memory efficient training techniques to reduce the memory footprint of storing intermediate tensors and data exchanges (communications) across accelerators, while ensuring high processing elements (PE) utilization.

In this survey, we review the generic techniques that boost computation and memory efficiency for training attention-based models, *i.e.*, Transformers, as shown in Figure 1. We characterize them by the technical innovations and primary use case, summarize them and draw connections between them. We are primarily interested in arithmetic innovations that improve the training efficiency of Transformers and also briefly discuss hardware/algorithm codesign advances. We leave the review of hardware accelerator design, a broad class, as future work.

2 Computation Efficiency

2.1 Optimization

Optimizer. To achieve a faster convergence rate for gradient descent, a classic solution is to fuse the momentum technique, where each step is a combination of the steepest descent direction and the most recent iterate displacement, helping to accelerate gradient descent in the relevant direction and dampens oscillations. The seminal works include Nesterov's accelerated gradient [Nesterov, 1983] for convex optimization and proximal gradient with momentum [Li *et al.*, 2017] towards non-convex problems, etc. To meet the demand of large-scale optimization of machine learning mod-

[†]Correspondence should be addressed to BZ.
[1]https://lambdalabs.com/blog/demystifying-gpt-3/

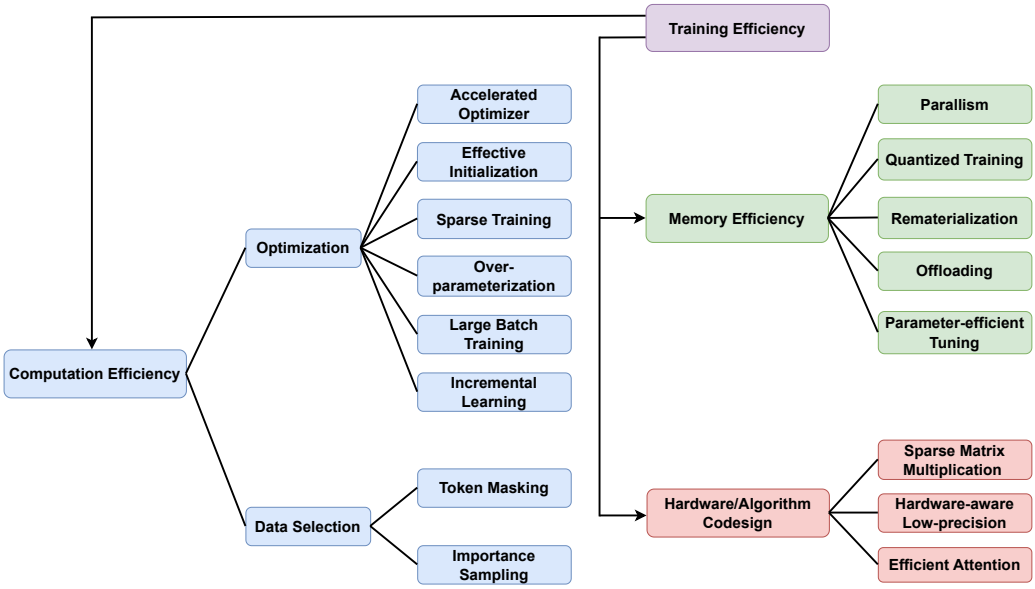

Figure 1: Overview of the efficient training of Transformers including computation efficiency, memory efficiency and hardware/algorithm codesign perspectives.

els, dominant optimizers are designed in a stochastic fashion. In particular, stochastic gradient descent (SGD) with momentum and the adaptive learning rate estimation method Adam [Kingma and Ba, 2015] are widely used to train deep neural networks. Empirically, training Transformers with Adam outperforms the SGD counterpart, and [Zhang *et al.*, 2019b] demystifies that a heavy-tailed distribution of the noise in stochastic gradients is the main cause of SGD's poor performance and understands Adam through the lens of adaptive noise clipping. By default, AdamW [Loshchilov and Hutter, 2019], a variant of Adam which decouples the L_2 regularization and the weight decay, is the most widely used optimizer for Transformers. More recently, Google searches optimization algorithms and discovers a simple and effective optimizer called Lion [Chen *et al.*, 2023]. Lion only keeps track of the momentum with the first-order gradient, and its update only considers the sign direction and has the same magnitude for each parameter, which is very different from the adaptive optimizers like AdamW. In practice, Lion in general converges faster, and is more memory-efficient and accurate than AdamW for training Transformers on various benchmarks. We refer readers to [Lin *et al.*, 2020b; Bottou *et al.*, 2018] for more details about accelerated optimization methods in machine learning.

To improve the generalization of Transformers, Sharpness-aware minimization (SAM) [Foret *et al.*, 2021] seeks to simultaneously minimize loss value and loss sharpness, based on the connection between the geometry of the loss landscape and generalization, *i.e.*, a flatter minimum tends to improve generalization. The following work [Chen *et al.*, 2022] applies SAM to Transformer, observing significant accuracy gains via smoothing the loss surface. However, SAM needs to solve a bi-level min-max optimization problem, which nearly doubles the training time. To accelerate optimization, [Du *et al.*, 2022a] proposes stochastic weight perturbation to preserve the generalization capability and sharpness-sensitive subset selection strategies. More recently, [Du *et al.*, 2022b] designs a near zero-cost proxy of the sharpness loss by replacing the sharpness estimation as the KL-divergence between the two consecutive update steps.

Initialization. A good initialization is essential to stabilize training, enable higher learning rate, accelerate convergence, and improve generalization. Thus, many works have been proposed for better initialization of Transformers. Specifically, Fixup [Zhang *et al.*, 2019a] proposes to properly rescale a standard initialization to ensure proper gradient norm to avoid exploding or vanishing gradients, which can train very deep networks with over 10,000 layers without adding normalization layers. Based on the insight that the function computed by normalized residual blocks is close to the identity function (*i.e.*, unit variance), the following works ReZero [Bachlechner *et al.*, 2021] and SkipInit [De and Smith, 2020] simply initialize each layer to perform the identity operation. Specifically, they add a learnable scaling multiplier on the output of each residual block:

$$\mathbf{x}_{l+1} = \mathbf{x}_l + \alpha_l F_l(\mathbf{x}_l), \tag{1}$$

where $\mathbf{x}_l$ and $F_l(\cdot)$ are the input and the function at layer l, where the function can be multi-head self-attention layers (MSA) and feed-forward networks (FFN), and α_l is simply initialized to 0. Customized to Transformers, T-Fixup [Huang *et al.*, 2020] analyzes that part of the optimization difficulty comes from the unstable early updates in the Adam optimizer as the variance of the second-order momentum is unbounded. Therefore, it follows Fixup to adopt rescaling schemes for the initialization of residual blocks. All the above-mentioned methods remove batch/layer normalization from all blocks and train without learning rate warmup. On training deep vision Transformers (ViT), [Touvron *et al.*, 2021a] proposes channel-wise learnable scaling factors and empirically ob-

serve that re-introducing the warmup and layer normalization techniques can make training more stable.

Apart from the rescaling paradigm, some literature proposes to improve initialization from a new perspective by leveraging the relationship between self-attention and convolutional layers. [Cordonnier *et al.*, 2020] proves that a multi-head self-attention layer with N heads and a relative positional encoding with dimension $D \geq 3$ can be reparameterized to express any convolutional layer of filter size $\sqrt{N} \times \sqrt{N}$. The attention can be decomposed into a content term and a (relative) positional term, where the latter determines the center and width of attention of each head. Based on this property, ConViT [d'Ascoli *et al.*, 2021] learns to control the locality by adding a soft gating parameter to balance the two terms, which has the effect of incorporating soft convolutional inductive biases into global self-attention.

Sparse training. The key idea of sparse training is to directly train sparse subnetworks instead of the full networks from scratch without sacrificing accuracy. The reliability was first demonstrated by the lottery ticket hypothesis (LTH) [Frankle and Carbin, 2019] that a dense, randomly initialized network contains subnetworks (winning tickets) which can be trained in isolation to match the accuracy of the original network. However, LTH requires identifying the winning tickets in an alternating train-prune-retrain manner, which makes the training extremely costly for large models and datasets, limiting the practical benefits. In light of this, follow-up works with higher training efficiency can be roughly categorized into three categories: (i) find sparse networks once at initialization by measuring the importance of connections on the loss, eliminating the need for the complex iterative optimization schedule [Lee *et al.*, 2019; Wang *et al.*, 2020] ; (ii) identify the winning tickets in Transformers at a very early training stage via low-cost schemes and then merely train these early tickets until convergence [You *et al.*, 2020; Chen *et al.*, 2021d]; (iii) use an alternating pruning and growing schedule to dynamically update model sparsity patterns throughout training, suitable for general architectures [Evci *et al.*, 2020; Chen *et al.*, 2021c].

Overparameterization. Practical DNNs are heavily overparameterized, where the number of learnable parameters is much larger than the number of training samples. It is observed that overparameterization empirically improves both convergence and generalization, with theoretical guarantee though not sufficient. The early work [Arora *et al.*, 2018] mathematically proves that increasing depth as overparameterization in linear neural networks can accelerate SGD convergence. [Li and Liang, 2018] further explores two-layer non-linear neural networks and [Allen-Zhu *et al.*, 2019b] proves that SGD can converge to global minima on the training objective of DNNs in *polynomial time*, assuming training samples are not duplicated, and the number of parameters is polynomial in the number of training samples and network depth. In terms of generalization, [Allen-Zhu *et al.*, 2019a] theoretically proves that a sufficiently overparameterized (three-layer) neural network generalizes to the population risk and an intriguing property is that *there exists an accurate network in the close neighborhood of any point on the SGD training trajectory with high probability over random initialization.* Note that it has deep connections with LTH as it partially explains why LTH stands in sparse training as good small sub-networks with low risks are plentiful due to overparameterization. Applied to Transformers, [Li *et al.*, 2020] exploits the faster convergence and better generalization from the overparameterization theory to design an efficient training pipeline: training a very large model, then perform early stopping and heavily compress it, analogous to LTH.

Large batch training. Another prevailing way to accelerate training is to use a large batch size, delivering a reduced number of iterations per epoch and better computing resource utilization. From the statistical view, large batch training reduces the variance of the stochastic gradient estimates, so a reliable step size needs to be tuned for better convergence [Bottou *et al.*, 2018]. At the era of convolutional neural networks, [Goyal *et al.*, 2017] employs the linear scaling of the learning rate to train ResNet-50 on ImageNet with a batch size of 8,192 in 1 hour. More advanced step size estimation methods are then proposed. The widely used methods are LARS [You *et al.*, 2017] for SGD and LAMB [You *et al.*, 2019] for Adam, which propose to use layerwise adaptive learning rates for ResNet and Transformers respectively. The layerwise adaptation strategy can be formulated as

$$\mathbf{w}_{t+1}^i = \mathbf{w}_t^i - \eta_t \frac{\phi(\|\mathbf{w}_t^i\|)}{\|\boldsymbol{\gamma}_t^i\|} \boldsymbol{\gamma}_t^i, \tag{2}$$

where η_t, $\mathbf{w}_t^i$ and $\boldsymbol{\gamma}_t^i$ are the learning rate, parameters and the momentum-based gradients of the i-th layer at time step t, ϕ is a scaling function. It equips with a normalization term that provides robustness to exploding gradients and plateaus, and the scaling term ensures that the norm of the update is of the same order as that of the parameter, promoting faster convergence. More recently, more powerful optimization methods customized to large batch training have been empirically shown to perform well. For example, [Kaddour, 2022] shows that averaging the weights of a certain number of latest checkpoints can facilitate faster training. DeepMind in [Hoffmann *et al.*, 2022] trains over 400 Transformer language models with varying scales of model size and # of training tokens, reaching to a practical hypothesis that the model size and the number of training tokens should be scaled equally for compute-optimal LLM training.

Incremental learning. The high-level concept of incremental learning is relaxing the original challenging optimization problem into a sequence of easy-to-optimize subproblems, where the solution of one sub-problem can serve as a good initialization to the subsequent one to circumvent the training difficulty, in analogy with annealing. Some works [Gong *et al.*, 2019; Gu *et al.*, 2021] propose to accelerate BERT pretraining by progressively stacking layers, properly initializing a larger model from a smaller one. [Zhang and He, 2020] goes in a reverse direction to train Transformers with stochastic depth via layer dropping, where it progressively increases dropping rate along both time dimension and depth dimension. Customized to ViT, AutoProg [Li *et al.*, 2022a] proposes to automatically decide whether, where and

how much should the model grow during progressive learning using neural architecture search. A key observation is that progressively increasing the resolution of the input images (reducing the patch size) can significantly accelerate ViT training, aligning with the widely known training dynamics that focus on low-frequency structure in the early stage and high-frequency semantics in the latter stage.

2.2 Data Selection

Apart from the model efficiency, data efficiency is also a crucial factor of efficient training.

Token masking. Token masking is a dominant approach in self-supervised pre-training tasks, such as masked language modeling (MLM) [Kenton and Toutanova, 2019; Brown *et al.*, 2020] and masked image modeling (MIM) [Bao *et al.*, 2022; He *et al.*, 2022]. The spirit of token masking is to randomly mask some input tokens and train the model to predict the missing content, e.g., vocabulary id or pixels, with the context information from the visible tokens. Since squeezing the sequence length reduces both the computational and memory complexity quadratically, skipping processing the masked tokens brings considerable training efficiency gain for MLM and MIM. For MLM, [Song *et al.*, 2019] proposes to jointly pre-train the encoder and decoder for language generation tasks while removing the masked tokens in the decoder to save memory and computation costs. For MIM, representative work [He *et al.*, 2022] shows that in vision, removing the masked image patches before the encoder demonstrates stronger performance and $3\times$ or more lower overall pre-training time and memory consumption than keeping the masked tokens. A similar phenomenon is also found in [Li *et al.*, 2022b] that for language-image pre-training, randomly masking and removing the masked image patches shows $3.7\times$ faster overall pre-training time than the original CLIP [Radford *et al.*, 2021].

Importance sampling. Importance sampling over data, also known as data pruning, is theoretically guaranteed to accelerate stochastic gradient algorithms for supervised learning by prioritizing informative training examples, mainly benefiting from variance reduction. For DNNs, a principal way of estimating per-sample importance is to use gradient norm, and [Katharopoulos and Fleuret, 2018; Johnson and Guestrin, 2018] use different approximations to make calculating these norms tractable. [Paul *et al.*, 2021] further speeds up the sampling process similar to the early-bird LTH, but in the data domain, that simple average gradient norms or error ℓ_2-norms over several weight initializations can be used to identify important examples at the very early stage in training. More recently, [Sorscher *et al.*, 2022] shows an exciting analytic theory that *the scaling of test error with dataset size can break beyond power scaling laws and be reduced to at least exponential scaling if equipped with a superior data pruning metric*, and it employs a self-supervised metric using k-means clustering. It demonstrates a promising direction towards more efficient neural scaling laws based on data importance sampling.

Method	Class
Micikevicius *et al.* [Micikevicius *et al.*, 2018]	AMP
Chen *et al.* [Chen *et al.*, 2016]	Rematerialization
Herrmann *et al.* [Herrmann *et al.*, 2019]	Rematerialization
ZeRO-Offload [Ren *et al.*, 2021]	Offloading
Beaumont *et al.* [Beaumont *et al.*, 2021]	Offloading + Rematerization
ZeRO [Rajbhandari *et al.*, 2020]	DP+MP+AMP
Megatron-LM [Shoeybi *et al.*, 2019]	DP+TP
GPipe [Huang *et al.*, 2019]	DP+PP
torchgpipe [Kim *et al.*, 2020]	PP+Rematerization
Megatron-LM*[Narayanan *et al.*, 2021]	DP+TP+PP+AMP
Wang *et al.* [Wang *et al.*, 2018]	FP8 Training
Cambier *et al.* [Cambier *et al.*, 2020]	FP8 Training
Mesa [Pan *et al.*, 2021]	8-bit ACT
ACTNN [Chen *et al.*, 2021a]	2-bit ACT
GACT [Liu *et al.*, 2022]	2-bit ACT
[Lester *et al.*, 2021], [Jia *et al.*, 2022], [Houlsby *et al.*, 2019]	Addition-based PET
Bitfit [Zaken *et al.*, 2022], LoRA [Hu *et al.*, 2022]	Reparameterization-based PET

Table 1: Summary of memory efficient training methods. Abbreviations include: AMP= Automatic Mixed Precision, DP = Data Parallelism, MP = Model Parallelism, TP = Tensor Parallelism, PP = Pipeline Parallelism, ACT = Activation Compressed Training and PET = Parameter-efficient Tuning.

3 Memory Efficiency

Apart from the computation burden, the growing model size of large Transformer models, *e.g.*, from BERT [Kenton and Toutanova, 2019] 345M parameter model to GPT-3 of 1.75 trillion parameters, is a key bottleneck for training as they do not fit into the memory of a single device. We first analyze the memory consumption of the existing model training frameworks, which is occupied by 1) *model states*, including optimizer states (*e.g.*, momentum and variance in Adam), gradients and parameters; and 2) *activations* (we ignore temporary buffers and idle fragmented memory as they are relatively small). We summarize the memory efficient training methods in Table 1. In the following, we discuss dominant solutions to optimize memory usage.

Parallelism. Training large DNNs with parallelism across devices is a common practice to meet the memory demands. There are basically two paradigms: *Data Parallelism* (DP) which distributes a minibatch of data across different devices and *Model Parallelism* (MP) which allocates subgraphs of a model across multiple workers. For DP, with the increase of available workers, the batch size is close to linear scaling. Large batch training discussed in Sec. 2 is developed for this case. However, it is obvious that DP has high communication/computation efficiency but poor memory efficiency - when model becomes large, the single device cannot store the model replica and the synchronized communications for gradients can hinder the scalability of DP. Therefore, DP itself is only suitable for training small to moderate models. To improve the scalability of DP, one solution for Transformer is parameter sharing [Lan *et al.*, 2020], known as Albert, but it limits the representational power. More recently, ZeRO [Rajbhandari *et al.*, 2020] incorporates uniform partitioning strategy with DP, where each data parallel process merely deals with one partition of model states, working in a mixed precision regime. To deal with very large DNNs, one always need to utilize model parallelism to allocate different layers across multiple accelerators in a "vertical" man-

ner. Though MP has good memory efficiency, its communication and computation efficiency is low due to high volume data transfer cross devices and poor PE utilization. Luckily, there are two strategies to further boost MP efficiency in an orthogonal "horizontal" dimension, including *Tensor Parallelism* (TP) and *Pipeline Parallelism* (PP). TP partitions a tensor operation in a layer across workers for faster computation and more memory saving. Customized to Transformer-based models, Megatron-LM [Shoeybi *et al.*, 2019] slices both MSA and FFN across GPUs and requires only a few extra All-Reduce operations in the forward and backward pass, allowing them to train models up to 8.3 billion parameters using 512 GPUs. In terms of PP, it was originally proposed in GPipe [Huang *et al.*, 2019], which splits the input mini-batch into multiple smaller micro-batches, enabling different accelerators (sequential layers are partitioned across accelerators) to work on different micro-batches simultaneously before applying a single synchronous gradient update for the entire mini-batch. However, it still suffers from pipeline bubbles (accelerator idle time) that reduce efficiency. In particular, PyTorch implements the torchgpipe [Kim *et al.*, 2020], which performs micro-batch PP with checkpointing, allowing scaling to a large number of micro-batches to minimize the bubble overhead.

Note that DP and MP are orthogonal and so one can use both simultaneously to train larger models with higher computation and memory capacity. For example, Megatron-LM* [Narayanan *et al.*, 2021] and DeepSpeed [Rasley *et al.*, 2020] compose tensor, pipeline, and data parallelism to scale training to thousands of GPUs.

Quantized training. The standard routine for training neural networks adopts full-precision (*i.e.*, FP32). In contrast, quantized training trains neural networks from scratch in reduced precision by compressing the activations/weights/gradients into low-bit values (*e.g.*, FP16 or INT8). It has been shown in previous works that reduced precision training [Zhou *et al.*, 2016; Hubara *et al.*, 2017] can accelerate neural network training with favorable performance. For Transformers, the most widely adopted approach is automatic mixed-precision (AMP) training [Micikevicius *et al.*, 2018]. Specifically, AMP stores a master copy of weights in full-precision for updates while the activations, gradients and weights are stored in FP16 for arithmetic. Compared to full-precision training, AMP is able to achieve faster training/inference speed and reduce memory consumption during network training. For example, based on a batch size of 64 and image resolution of 224×224, training a DeiT-B [Touvron *et al.*, 2021b] on RTX3090 under AMP is $2\times$ faster than full-precision training (305 *vs.* 124 images/s), as well as consuming 22% less peak GPU memory (7.9GB *vs.* 10.2GB). While it is commonly believed that at least 16-bits is necessary to train networks without impacting model accuracy [Micikevicius *et al.*, 2018; Das *et al.*, 2018], the most recent support for FP8 training on NVIDIA H100 has shown promising results on Transformer training, where training DeiT-S and GPT [Brown *et al.*, 2020] under FP8 can match those of 16-bit training. Apart from reduced precision training which simultaneously quantizes activations/weights/gradients, activation compressed training (ACT) [Chen *et al.*, 2021a] stores low-precision approximate copies of activations while computing the forward pass exactly, which helps to reduce the overall memory consumption during training. The saved activations are then dequantized to the original precision in the backward pass to calculate gradients. Recent work [Pan *et al.*, 2021; Liu *et al.*, 2022] further propose to customize ACT to support memory-efficient Transformer training.

Rematerialization and offloading. *Rematerialization*, also known as *checkpointing* [Chen *et al.*, 2016], is a widely used technique for space-time tradeoff that only stores a portion of activations/weights during the forward pass and recomputes the rest during the backward pass. [Chen *et al.*, 2016] provides a simple periodic schedule which was implemented in PyTorch[2], but it is only optimal for homogeneous sequential networks. More advanced methods such as [Herrmann *et al.*, 2019] implements optimal checkpointing for heterogeneous networks[3]. In terms of *offloading*, it is a technique to use external memory such as CPU memory as an extension of GPU memory to increase memory capacity during training, through communications between GPU and CPU. The model states as well as activations, can be offloaded to CPU, but the optimal choice needs to minimize communication cost (*i.e.*, data movement) to/from GPU, reduce CPU computation and maximize GPU memory saving. A representative work is ZeRO-Offload [Ren *et al.*, 2021], which offers optimal offloading strategy customized to mixed-precision training with Adam optimizer. It offloads all fp32 model states and the fp16 gradients on the CPU memory, and computes the fp32 parameter updates on CPU. The fp16 parameters are kept on GPU and the forward and backward computations are on GPU. For the best of both worlds, [Beaumont *et al.*, 2021] proposes to jointly optimize activation offloading and rematerialization.

Parameter-efficient tuning. The public model zoo represented by HuggingFace, which contains rich pretrained models that are ready to be downloaded and executed, is contributing significantly to reductions in training costs. Efficient tuning these readily available models is becoming a prevailing way to drastically cut training costs. As a powerful alternative for the vanilla full fine-tuning, parameter-efficient tuning (PET) only updates a small number of additional parameters while freezing the pretrained model to significantly reduce the storage burden, which scales with dynamic deployment scenarios without the need to store a separate instance of model for each case. The general PET approaches can be categorized into *addition-based* methods and *reparameterization-based* methods. The former attaches additional trainable parameters to the pretrained model and only tune these parameters. For example, [Lester *et al.*, 2021; Jia *et al.*, 2022] add trainable parameters to the input space, and [Houlsby *et al.*, 2019] adds the adapter module twice to each Transformer block after the MSA and FFN. However, the extra parameters introduce additional computation

[2]https://pytorch.org/

[3]https://gitlab.inria.fr/hiepacs/rotor

and memory overhead during inference. To tackle this challenge, the latter proposes to tune parameters that are inherently in the model [Zaken *et al.*, 2022] or new parameters that can be reparameterized into the model [Hu *et al.*, 2022], thereby yielding no sacrifice on the inference efficiency. Inspired by the observation that large language pretrained models have low intrinsic dimension [Aghajanyan *et al.*, 2021], the representative work LoRA [Hu *et al.*, 2022] approximates the update of self-attention weights into two low-rank matrices, which can be merged into the pretrained weights during inference. Notably, one of the most recognized efforts for democratizing LLM is Stanford Alpaca [4], which is fine-tuned from the open-sourced LLaMA models [Touvron *et al.*, 2023] using the 52K instruction-following data generated from ChatGPT. To fine-tune it cheaply and efficiently, its variant Alpaca-LoRA [5] further adopts the low-rank LoRA to enable instruct-tuning LLaMA on customer hardware, showing training can be done within hours on a single RTX 4090.

Open-source frameworks. There are several widely adopted prototypes for training large Transformer models at scale, in which Microsoft DeepSpeed[6], HPC-AI Tech Colossal-AI[7] and Nvidia Megatron-LM[8] are the pioneering ones. Specifically, DeepSpeed is implemented mainly based on [Rasley *et al.*, 2020] and ZeRO series works [Rajbhandari *et al.*, 2020; Ren *et al.*, 2021], Colossal-AI is built upon [Bian *et al.*, 2021], and Megatron-LM implements [Narayanan *et al.*, 2021]. All of these support data and model parallelism in mixed precision, along with other general practices such as offloading and rematerialization. More libraries for efficient distributed training include but not limited to HuggingFace Transformers[9], MosaicML Composer[10], Baidu PaddlePaddle[11], Bytedance Lightseq[12], EleutherAI GPT-NeoX[13], etc.

4 Hardware/Algorithm Co-design

Apart from computing and memory burden, designing efficient hardware accelerators enables faster training and inference for DNNs. Specifically, compared with central processing units (CPUs), graphics processing units (GPUs) are more powerful to perform matrix multiplication due to the high degree of parallelism. For applications that focus on specific computation tasks, application-specific integrated circuits (ASICs) have the advantage of low power consumption, and high training/inference speed. For example, a tensor processing unit (TPU) designed by Google delivered 30∼80× higher performance-per-watt than contemporary CPUs and GPUs [Jouppi *et al.*, 2017]. However, ASICs are not easily reprogrammable or adaptable to a new task. In contrast, Field Programmable Gate Arrays (FPGAs) are designed to be reprogrammed to perform different functions as needed, and can also be used as a prototype for ASICs before finalizing the design. To further optimize the training efficiency of DNNs, especially Transformers, hardware-algorithm co-design takes the constraints and capabilities of the hardware into account when designing the algorithm, which will be introduced in the following subsections.

Sparse matrix multiplication. To reduce the computation overhead of Transformers, sparse general matrix multiplication (SpGEMM), which involves multiplying a sparse matrix with a dense matrix, takes advantage of the sparsity of the attention matrices to reduce the number of computations. There are several popular sparse matrix computation libraries, such as Intel Math Kernel Library[14] on CPU and cuSPARSE[15], CUSP[16] and 2:4 structured sparsity [17][Mishra *et al.*, 2021] on GPU. However, due to the irregular sparsity, SpGEMM is often hardware unfriendly to general-purpose processors, such as CPUs and GPUs. To handle this, specialized hardware accelerators, such as FPGAs and ASICs, are required to handle the poor data locality issue. For example, OuterSPACE [Pal *et al.*, 2018] transforms matrix multiplication into an outer product procedure and eliminates redundant memory accesses by decoupling multiplication from accumulation. To take full advantage of this reduction without introducing significant overheads, OuterSPACE builds a custom accelerator with reconfigurable memory hierarchy and achieves a mean speedup of 7.9× over the CPU running Intel Math Kernel Library and 14.0× against the GPU running CUSP. Furthermore, to alleviate the data movement bottleneck caused by high sparsity, ViTCoD [You *et al.*, 2023] uses a learnable auto-encoder to compress the sparse attentions to a much more compact representation and designs encoder and decoder engines to boost the hardware utilization.

Hardware-aware low-precision. Lowering the precision of the computations reduces the amount of memory and computation, which can be implemented in hardware-friendly fixed-point or integer representations instead of floating-point ones. As a result, we can use lower precision multipliers, adders, and memory blocks, resulting in a significant improvement in power consumption and speedup. Moreover, low-precision arithmetic can be combined with other techniques, such as pruning and low-rank approximation, to achieve further acceleration. For example, Sanger [Lu *et al.*, 2021] uses 4-bit queries and keys to compute the quantized prediction of sparse attention matrix. Then, the sparse attention masks are rearranged into structured blocks and handled by reconfigurable hardware. The following work DOTA [Qu *et al.*, 2022] identifies unimportant connections in attention using low-rank transformation and low-precision computation. By incorporating token-level parallelism and out-of-order execution, DOTA achieves a 152.6× speedup over GPU.

[4] https://github.com/tatsu-lab/stanford_alpaca
[5] https://github.com/tloen/alpaca-lora
[6] https://github.com/microsoft/DeepSpeed
[7] https://github.com/hpcaitech/ColossalAI
[8] https://github.com/NVIDIA/Megatron-LM
[9] https://github.com/huggingface/transformers
[10] https://github.com/mosaicml/composer
[11] https://github.com/PaddlePaddle/Paddle
[12] https://github.com/bytedance/lightseq
[13] https://github.com/EleutherAI/gpt-neox
[14] https://www.intel.com/content/www/us/en/developer/tools/oneapi/onemkl.html
[15] https://docs.nvidia.com/cuda/cusparse/
[16] https://cusplibrary.github.io/
[17] https://developer.nvidia.com/blog/accelerating-inference-with-sparsity-using-ampere-and-tensorrt/

Efficient attention. Apart from the sparse matrix multiplication and low-precision computation, several pioneering works focus on efficient and lightweight attention implementation in hardware [Ham *et al.*, 2020; Ham *et al.*, 2021; Dao *et al.*, 2022]. Specifically, A^3 [Ham *et al.*, 2020] only selects those keys that are likely to have high similarity with the given queries to reduce the amount of computation in attention. ELSA [Ham *et al.*, 2021] filters out irrelevant keys for a particular query based on the hashing similarity to save computation. With an efficient hardware accelerator, ELSA achieves a speedup of $58.1\times$ as well as three orders of magnitude improvements in energy efficiency compared to an Nvidia V100 GPU equipped with 16GB memory. Notably, FlashAttention [Dao *et al.*, 2022] proposes to exploit tiling to reduce the I/O communication between GPU high bandwidth memory (HBM) and on-chip SRAM, which is becoming a default fast and memory-efficient attention module for speedup.

5 Conclusion and Future Research

In this survey, we have reviewed several important factors that improve the training of Transformers: 1) appropriate initialization and optimization paradigms that can accelerate the convergence rate with fewer training iterations, resulting in *lower computational costs*; 2) higher data efficiency by sampling informative training samples towards *more efficient neural scaling laws* of test error with respect to dataset size; 3) memory-efficient techniques to meet the memory requirements for training large Transformers, which requires *jointly optimizing PE utilization, memory and communication footprints across accelerators*, using parallelism, low-precision arithmetic, checkpointing and offloading strategies, etc; 4) *hardware and algorithm co-design* to maximize the training scalability on hardware platforms.

We finally highlight several promising directions based on the reviewed progress: (i) joint training and deployment efficiency optimization. In reality, we usually need to deploy models to diverse tasks and platforms with different resource constraints. Therefore, it is highly desirable to develop new methods for efficiently training an elastic supertnet that supports many diverse architectural configurations following single-shot NAS [Chen *et al.*, 2021b] or mixture of experts [Fedus *et al.*, 2022], for multiple tasks and budget requirements; (ii) on-device training [Lin *et al.*, 2020a] on the edge with limited resources (*e.g.*, low battery and memory capacity), to avoid frequently transmitting data which results in privacy and latency issues; (iii) combine efficient approximation techniques such as token/model pruning, low-rank factorization, lightweight architecture design, dynamic neural networks and etc, to reduce the model size and computational cost in the complimentary sense; (iv) a standard benchmark to evaluate and compare the efficient training methods. Having such a benchmark would fasten the adoption of these methods and lead to actual costs reduction down the line.

References

[Aghajanyan *et al.*, 2021] Armen Aghajanyan, Sonal Gupta, et al. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL*, 2021.

[Allen-Zhu *et al.*, 2019a] Zeyuan Allen-Zhu, Yuanzhi Li, and Yingyu Liang. Learning and generalization in overparameterized neural networks, going beyond two layers. In *NeurIPS*, 2019.

[Allen-Zhu *et al.*, 2019b] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In *ICML*, 2019.

[Arora *et al.*, 2018] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. In *ICML*, 2018.

[Bachlechner *et al.*, 2021] Thomas Bachlechner, Bodhisattwa Prasad Majumder, et al. Rezero is all you need: Fast convergence at large depth. In *UAI*, 2021.

[Bao *et al.*, 2022] Hangbo Bao, Li Dong, et al. Beit: Bert pre-training of image transformers. In *ICLR*, 2022.

[Beaumont *et al.*, 2021] Olivier Beaumont, Eyraud-Dubois, et al. Efficient combination of rematerialization and offloading for training dnns. In *NeurIPS*, 2021.

[Bian *et al.*, 2021] Zhengda Bian, Hongxin Liu, et al. Colossal-ai: A unified deep learning system for large-scale parallel training. *arXiv preprint arXiv:2110.14883*, 2021.

[Bottou *et al.*, 2018] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 2018.

[Brown *et al.*, 2020] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

[Cambier *et al.*, 2020] Léopold Cambier, Anahita Bhiwandiwalla, et al. Shifted and squeezed 8-bit floating point format for low-precision training of deep neural networks. In *ICLR*, 2020.

[Chen *et al.*, 2016] Tianqi Chen, Bing Xu, et al. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[Chen *et al.*, 2021a] Jianfei Chen, Lianmin Zheng, et al. Actnn: Reducing training memory footprint via 2-bit activation compressed training. In *ICML*, 2021.

[Chen *et al.*, 2021b] Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. Autoformer: Searching transformers for visual recognition. In *ICCV*, 2021.

[Chen *et al.*, 2021c] Tianlong Chen, Yu Cheng, et al. Chasing sparsity in vision transformers: An end-to-end exploration. In *NeurIPS*, 2021.

[Chen *et al.*, 2021d] Xiaohan Chen, Yu Cheng, et al. Earlybert: Efficient bert training via early-bird lottery tickets. In *ACL*, 2021.

[Chen *et al.*, 2022] Xiangning Chen, Cho-Jui Hsieh, et al. When vision transformers outperform resnets without pretraining or strong data augmentations. In *ICLR*, 2022.

[Chen *et al.*, 2023] Xiangning Chen, Chen Liang, et al. Symbolic discovery of optimization algorithms. *arXiv preprint arXiv:2302.06675*, 2023.

[Cordonnier *et al.*, 2020] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers. In *ICLR*, 2020.

[Dao *et al.*, 2022] Tri Dao, Daniel Y. Fu, et al. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *NeurIPS*, 2022.

[Das *et al.*, 2018] Dipankar Das, Naveen Mellempudi, et al. Mixed precision training of convolutional neural networks using integer operations. In *ICLR*, 2018.

[De and Smith, 2020] Soham De and Sam Smith. Batch normalization biases residual blocks towards the identity function in deep networks. In *NeurIPS*, 2020.

[Du *et al.*, 2022a] Jiawei Du, Hanshu Yan, et al. Efficient sharpness-aware minimization for improved training of neural networks. In *ICLR*, 2022.

[Du *et al.*, 2022b] Jiawei Du, Daquan Zhou, et al. Sharpness-aware training for free. In *NeurIPS*, 2022.

[d'Ascoli *et al.*, 2021] Stéphane d'Ascoli, Hugo Touvron, et al. Convit: Improving vision transformers with soft convolutional inductive biases. In *ICML*, 2021.

[Evci *et al.*, 2020] Utku Evci, Trevor Gale, et al. Rigging the lottery: Making all tickets winners. In *ICML*, 2020.

[Fedus *et al.*, 2022] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *JMLR*, 2022.

[Foret *et al.*, 2021] Pierre Foret, Ariel Kleiner, et al. Sharpness-aware minimization for efficiently improving generalization. In *ICLR*, 2021.

[Frankle and Carbin, 2019] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *ICLR*, 2019.

[Gong *et al.*, 2019] Linyuan Gong, Di He, et al. Efficient training of bert by progressively stacking. In *ICML*, 2019.

[Goyal *et al.*, 2017] Priya Goyal, Piotr Dollár, et al. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

[Gu *et al.*, 2021] Xiaotao Gu, Liyuan Liu, et al. On the transformer growth for progressive bert training. In *NAACL*, 2021.

[Ham *et al.*, 2020] Tae Jun Ham, Sung Jun Jung, et al. Aˆ 3: Accelerating attention mechanisms in neural networks with approximation. In *HPCA*, 2020.

[Ham *et al.*, 2021] Tae Jun Ham, Yejin Lee, et al. Elsa: Hardware-software co-design for efficient, lightweight self-attention mechanism in neural networks. In *ISCA*, 2021.

[He *et al.*, 2022] Kaiming He, Xinlei Chen, et al. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

[Herrmann *et al.*, 2019] Julien Herrmann, Olivier Beaumont, et al. Optimal checkpointing for heterogeneous chains: how to train deep neural networks with limited memory. *arXiv preprint arXiv:1911.13214*, 2019.

[Hoffmann *et al.*, 2022] Jordan Hoffmann, Sebastian Borgeaud, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[Houlsby *et al.*, 2019] Neil Houlsby, Andrei Giurgiu, et al. Parameter-efficient transfer learning for nlp. In *ICML*, 2019.

[Hu *et al.*, 2022] Edward J Hu, yelong shen, et al. Lora: Low-rank adaptation of large language models. In *ICLR*, 2022.

[Huang *et al.*, 2019] Yanping Huang, Youlong Cheng, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In *NeurIPS*, 2019.

[Huang *et al.*, 2020] Xiao Shi Huang, Felipe Perez, et al. Improving transformer optimization through better initialization. In *ICML*, 2020.

[Hubara *et al.*, 2017] Itay Hubara, Matthieu Courbariaux, et al. Quantized neural networks: Training neural networks with low precision weights and activations. *JMLR*, 2017.

[Jia *et al.*, 2022] Menglin Jia, Luming Tang, Bor-Chun Chen, Claire Cardie, Serge Belongie, Bharath Hariharan, and Ser-Nam Lim. Visual prompt tuning. In *ECCV*, 2022.

[Johnson and Guestrin, 2018] Tyler B Johnson and Carlos Guestrin. Training deep models faster with robust, approximate importance sampling. In *NeurIPS*, 2018.

[Jouppi *et al.*, 2017] Norman P Jouppi, Cliff Young, et al. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, 2017.

[Kaddour, 2022] Jean Kaddour. Stop wasting my time! saving days of imagenet and bert training with latest weight averaging. In *NeurIPS workshop*, 2022.

[Katharopoulos and Fleuret, 2018] Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *ICML*, 2018.

[Kenton and Toutanova, 2019] Jacob Devlin Ming-Wei Chang Kenton and Lee Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[Kim *et al.*, 2020] Chiheon Kim, Heungsub Lee, et al. torchgpipe: On-the-fly pipeline parallelism for training giant models. *arXiv preprint arXiv:2004.09910*, 2020.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[Lan *et al.*, 2020] Zhenzhong Lan, Mingda Chen, et al. Albert: A lite bert for self-supervised learning of language representations. In *ICLR*, 2020.

[Lee *et al.*, 2019] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip Torr. Snip: Single-shot network pruning based on connection sensitivity. In *ICLR*, 2019.

[Lester *et al.*, 2021] Brian Lester, Rami Al-Rfou, and Noah Constant. The power of scale for parameter-efficient prompt tuning. In *EMNLP*, 2021.

[Li and Liang, 2018] Yuanzhi Li and Yingyu Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *NeurIPS*, 2018.

[Li *et al.*, 2017] Qunwei Li, Yi Zhou, et al. Convergence analysis of proximal gradient with momentum for nonconvex optimization. In *ICLR*, 2017.

[Li *et al.*, 2020] Zhuohan Li, Eric Wallace, et al. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *ICML*, 2020.

[Li *et al.*, 2022a] Changlin Li, Bohan Zhuang, et al. Automated progressive learning for efficient training of vision transformers. In *CVPR*, 2022.

[Li *et al.*, 2022b] Yanghao Li, Haoqi Fan, et al. Scaling language-image pre-training via masking. *arXiv preprint arXiv:2212.00794*, 2022.

[Lin *et al.*, 2020a] Ji Lin, Wei-Ming Chen, et al. Mcunet: Tiny deep learning on iot devices. In *NeurIPS*, 2020.

[Lin *et al.*, 2020b] Zhouchen Lin, Huan Li, and Cong Fang. Accelerated optimization for machine learning. *Nature Singapore: Springer*, 2020.

[Liu *et al.*, 2022] Xiaoxuan Liu, Lianmin Zheng, et al. GACT: activation compressed training for generic network architectures. In *ICML*, 2022.

[Loshchilov and Hutter, 2019] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

[Lu *et al.*, 2021] Liqiang Lu, Yicheng Jin, et al. Sanger: A co-design framework for enabling sparse attention using reconfigurable architecture. In *MICRO*, 2021.

[Micikevicius *et al.*, 2018] Paulius Micikevicius, Sharan Narang, et al. Mixed precision training. In *ICLR*, 2018.

[Mishra *et al.*, 2021] Asit Mishra, Jorge Albericio Latorre, et al. Accelerating sparse deep neural networks. *arXiv preprint arXiv:2104.08378*, 2021.

[Narayanan *et al.*, 2021] Deepak Narayanan, Mohammad Shoeybi, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *SC*, 2021.

[Nesterov, 1983] Yurii Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady ANSSSR*, volume 269, pages 543–547, 1983.

[Pal *et al.*, 2018] Subhankar Pal, Jonathan Beaumont, et al. Outerspace: An outer product based sparse matrix multiplication accelerator. In *HPCA*, 2018.

[Pan *et al.*, 2021] Zizheng Pan, Peng Chen, et al. Mesa: A memory-saving training framework for transformers. *arXiv preprint arXiv:2111.11124*, 2021.

[Paul *et al.*, 2021] Mansheej Paul, Surya Ganguli, et al. Deep learning on a data diet: Finding important examples early in training. In *NeurIPS*, 2021.

[Qu *et al.*, 2022] Zheng Qu, Liu Liu, et al. Dota: detect and omit weak attentions for scalable transformer acceleration. In *ASPLOS*, 2022.

[Radford *et al.*, 2021] Alec Radford, Jong Wook Kim, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[Rajbhandari *et al.*, 2020] Samyam Rajbhandari, Jeff Rasley, et al. Zero: Memory optimizations toward training trillion parameter models. In *SC*, 2020.

[Rasley *et al.*, 2020] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *KDD*, 2020.

[Ren *et al.*, 2021] Jie Ren, Samyam Rajbhandari, et al. Zero-offload: Democratizing billion-scale model training. In *USENIX ATC*, 2021.

[Schwartz *et al.*, 2020] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 2020.

[Shoeybi *et al.*, 2019] Mohammad Shoeybi, Mostofa Patwary, et al. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

[Smith *et al.*, 2022] Shaden Smith, Mostofa Patwary, et al. Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv preprint arXiv:2201.11990*, 2022.

[So *et al.*, 2019] David So, Quoc Le, and Chen Liang. The evolved transformer. In *ICML*, pages 5877–5886, 2019.

[Song *et al.*, 2019] Kaitao Song, Xu Tan, et al. Mass: Masked sequence to sequence pre-training for language generation. In *ICML*, 2019.

[Sorscher *et al.*, 2022] Ben Sorscher, Robert Geirhos, et al. Beyond neural scaling laws: beating power law scaling via data pruning. In *NeurIPS*, 2022.

[Strubell *et al.*, 2019] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. In *ACM*, 2019.

[Touvron *et al.*, 2021a] Hugo Touvron, Matthieu Cord, et al. Going deeper with image transformers. In *ICCV*, 2021.

[Touvron *et al.*, 2021b] Hugo Touvron, Matthieu Cord, et al. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

[Touvron *et al.*, 2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[Vaswani *et al.*, 2017] Ashish Vaswani, Noam Shazeer, et al. Attention is all you need. In *NeurIPS*, 2017.

[Wang *et al.*, 2018] Naigang Wang, Jungwook Choi, et al. Training deep neural networks with 8-bit floating point numbers. In *NeurIPS*, 2018.

[Wang *et al.*, 2020] Chaoqi Wang, Guodong Zhang, and Roger Grosse. Picking winning tickets before training by preserving gradient flow. In *ICLR*, 2020.

[You *et al.*, 2017] Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

[You *et al.*, 2019] Yang You, Jing Li, et al. Large batch optimization for deep learning: Training bert in 76 minutes. In *ICLR*, 2019.

[You *et al.*, 2020] Haoran You, Chaojian Li, et al. Drawing early-bird tickets: Towards more efficient training of deep networks. In *ICLR*, 2020.

[You *et al.*, 2023] Haoran You, Zhanyi Sun, et al. Vitcod: Vision transformer acceleration via dedicated algorithm and accelerator co-design. In *HPCA*, 2023.

[Zaken *et al.*, 2022] Elad Ben Zaken, Yoav Goldberg, et al. Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *ACL*, 2022.

[Zhang and He, 2020] Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. In *NeurIPS*, 2020.

[Zhang *et al.*, 2019a] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. In *ICLR*, 2019.

[Zhang *et al.*, 2019b] Jingzhao Zhang, Sai Praneeth Karimireddy, et al. Why adam beats sgd for attention models. In *OpenReview*, 2019.

[Zhou *et al.*, 2016] Shuchang Zhou, Yuxin Wu, et al. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.