Hardware Software Co-design and Architectural Optimization of Deep Learning Models for Natural Language Processing

Thanakul Wattanawong
Kurt Keutzer

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-92
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-92.html

May 11, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Hardware Software Co-design and Architectural Optimization of Deep Learning Models for Natural Language Processing

Thanakul Wattanawong

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee

Professor Kurt Keutzer
Research Advisor

05 / 09 / 2023

(Date)

★ ★ ★ ★ ★ ★ ★

Sophia Shao

Assistant Professor Sophia Shao
Second Reader

05 / 11 / 2023

(Date)

Doc ID: 672c347797cf70b4547cff68f866d2b1e4048762

Abstract

Hardware Software Co-design and Architectural Optimization of Deep Learning Models for Natural Language Processing

by

Thanakul Wattanawong

Master of Science in Computer Science

University of California, Berkeley

Professor Kurt Keutzer, Advisor

Assistant Professor Sophia Shao, Co-chair

Transformer models are achieving state of the art performance across tasks in natural language processing, computer vision, and others. However, the amount of compute required to perform inference using Transformers has grown significantly over the past few years, making them unusable at the edge or in low-power electronics. Therefore, there is an increasing need to improve their efficiency with opportunities ranging from performing architectural modifications to designing domain-specific accelerators (DSA). In this work we present two approaches to optimizing the inference of Transformer models. One is a Hardware Software Co-design approach that jointly optimizes the hardware architecture alongside the Transformer architecture. We present a framework based on Neural Architecture Search (NAS) and evolutionary search that practitioners may use to find the best-matched hardware configuration and Transformer architecture that satisfy the required performance criteria. We optimize the Transformer for both inference latency and power consumption using a metric called Energy-Delay Product (EDP), and find that the framework can attain a 2.2× EDP improvement while tolerating a 0.1 point perplexity degradation, and 10.6× with a 1 point degradation over the baseline. The survey [19] that this work contributed to further combined other improvements such as insights from [20] for an overall 88.7× speedup. Based on the insights gained from the survey paper, we were able to conduct experiments on architectural optimizations on feedforward networks for small Transformers that are designed to reduce inference FLOPs and energy consumption. We find that the feedforward network, which accounts for roughly 60% of the parameter count and inference FLOPs of the model T5-Mini, can be removed with only a 2.7 point loss on MNLI-mm, a standard natural language inference benchmark. Along with a number of other ablations, we find that structural weight reparametrization can be used to reduce inference FLOPs and parameters by about 30% with only a one point drop on MNLI-mm.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank everyone in the PALLAS Lab and those on the Efficient NLP project for their advice and assistance along the way. In no particular order, I'd like to thank Sehoon Kim, Dr. Amir Gholami, Nicholas Lee, Sheng Shen, Xiuyu Li, Karttikeya Mangalam and Prof. Kurt Keutzer.

I would like to thank all my collaborators on the Hardware Software Co-design project including Coleman Hooper, Ruohan Yan, Hasan Genc, Dr. Qijing Huang, Grace Dinh, and Minwoo Kang.

The Hardware Software Co-design project is supported by a generous grant from a 2022 grant for AI System Hardware/Software Codesign research from Meta.

I would like to acknowledge the generous allocation of TPUs from the TPU Research Cloud program, which has made the computation for this project feasible.

Chapter 1

Introduction

In recent years, the Transformer architecture [35] has revolutionized the field of natural language processing. Transformers are characterized by their use of the attention mechanism, which allows the model to learn to focus on specific tokens in the input that are important for making predictions or generating new output.

However, as these models have grown, their compute requirements have also outpaced the hardware available to run them. Most modern deep learning models are currently run on either GPUs (Graphics Processing Unit) or specialized accelerators such as the TPU (Tensor Processing Unit), and while flagship model sizes have been growing at the rate of 240x every two years, the amount of memory available on accelerators has only grown by about 2x every two years [14]. This makes the cost of training and inference untenable for all but the most well-funded organizations, and thus we must find ways to create more efficient Transformers, especially in inference as that is where the majority of their lifetime compute will come from.

Although alternative scale-out techniques such as model parallelism have been proposed to alleviate this issue, here we focus on designing efficient Transformers for the single node case. In particular, I focus on two main directions. One is using Hardware Software Co-design to design an efficient Transformer pairing with an accelerator configuration. The other is optimizing the Transformer architecture directly for efficient inference.

Chapter 2

Related Works

2.1 The Transformer Architecture

The Transformer is characterized by its use of the attention mechanism [35], and models based on this architecture have become the backbone of modern natural language processing as they excel in tasks ranging from Natural Language Understanding, to translation, to generating human-like text. For example, BERT [11], an encoder-only model, achieved state of the art on Natural Language Understanding tasks such as GLUE [36] and SQuAD [27]. Other variants such as the encoder-decoder T5 model [26] further improved performance on these language understanding tasks by converting all problems to a text-to-text pretraining objective. More recently, the GPT family of Transformers [24, 25, 3, 22] have shown superior performance in generating human-like text, invigorating the field of generative AI research.

2.2 Efficient Transformer Architectures

The high compute costs of Transformers have motivated many works attempting to reduce it's cost. There are a multitude of works that focus on reducing the cost of self-attention [21, 2, 43, 31, 7, 32, 40, 38, 17, 9], which grows quadratically with the sequence length due to the all-to-all computation required. A minority of works focus on improving the cost of the feedforward network. In our survey paper [19], we have found that in smaller networks and smaller sequence lengths, the feedforward network dominates the overall cost, and so we have chosen to focus on this area.

For a more thorough survey of efficient Transformers refer to [29].

Chapter 3

Hardware Software Co-design

3.1 Overview

While general purpose computing devices such as CPUs and GPGPUs have been commonplace in deep learning for a while, modern specialized accelerators are becoming more popular due to their ability to perform the required operations needed in a more efficient manner. Additionally, the rich design space offered by technologies such as Chipyard [1] and Gemmini [13] have now enabled the ability to co-design a well-matched pairing of hardware alongside the software in order to achieve high efficiency.

The survey paper [19] that this work contributed to analyzed the runtime characteristics of Transformers, surveyed methodologies for designing efficient transformers, and finally performed a case study to apply the studied methods using Gemmini, the full-stack deep neural network accelerator generator. This work specifically focuses on the Neural Architecture Search (NAS) implenmentation and its merits. Sehoon Kim contributed to much of the implementation of the NAS framework. In the survey paper, others contributed performance increases that total up to 88.7×.

3.2 Key Findings

First I summarize the key findings that we discovered with regards to both the Gemmini accelerator and Transformer inference.

To set the background, other parts of the survey paper [19] found that:

- Gemmini, which was originally designed and benchmarked on CNN workloads, is not well suited for Transformer inference. In fact, due to the requirement to offload floating-point non-linear operations as well as (de)quantization operations, hardware utilization could be less than 1%.
- By using I-BERT for integer-only quantization and implementing dedicated acceleration units for nonlinear operations, a 39.6× improvement.

- Despite the fact that matrix multiplication schedules in Transformers only require three loops, they can be challenging to schedule, with the difference between the best and worst solutions being up to four orders of magnitude.

The contributions of this work and to the survey paper are as follows:

- By running our Neural Architecture Search framework over several hardware configurations, we found that Transformers benefit from having a large accumulator and smaller scratchpad size, whereas the opposite is more optimal for CNNs.
- Our Neural Architecture Search (NAS) can obtain a 2.24x EDP reduction with only 0.1 point perplexity drop and a 10.56x EDP reduction with only 1 point perplexity drop on the WikiText-2M dataset.

3.3 Codesign Methodology

Approach

As a case study, we applied our findings on the Gemmini [13] deep neural network accelerator platform in order to find a well-matched Transformer and hardware configuration. Although there are many previous works on designing deep learning architectures that perform well on a specific hardware ([39, 4, 5] among many others), most often assume the underlying hardware is fixed. Gemmini and the Chipyard infrastructure are able to simulate many hardware configurations, and so we have chosen to take advantage of that fact.

The Neural Architecture Search framework is summarized in Fig. 3.1 from [19]:

At a high level, we train a supernet that we can sample architectures from, and then conduct an evolutionary search process to find pareto-optimal architectures. The advantage of our method is the supernet only has to be trained once as it can take time, while the second phase only has to take place once given a set of lookup tables for delay and energy.

Experimental Setup

We use Neural Architecture Search (NAS) and adopt a BigNAS-style [42] strategy to train a supernet that we can draw random architectures from. Then, we use an evolutionary algorithm to search for pareto-optimal sub-networks out of the fully trained supernet. The NAS search space is composed of various combinations of the number of layers, number of heads, hidden dimension, and FFN dimension as described in Tab. 3.1.

Our baseline is a Language Modeling task and we train a randomly initialized 6-layer Transformer model on the WikiText-2 [33] benchmark using the biggest architecture possible in our NAS search space. We optimize for both latency and energy using simulated values from Timeloop [23] which were contributed by Minwoo Kang.

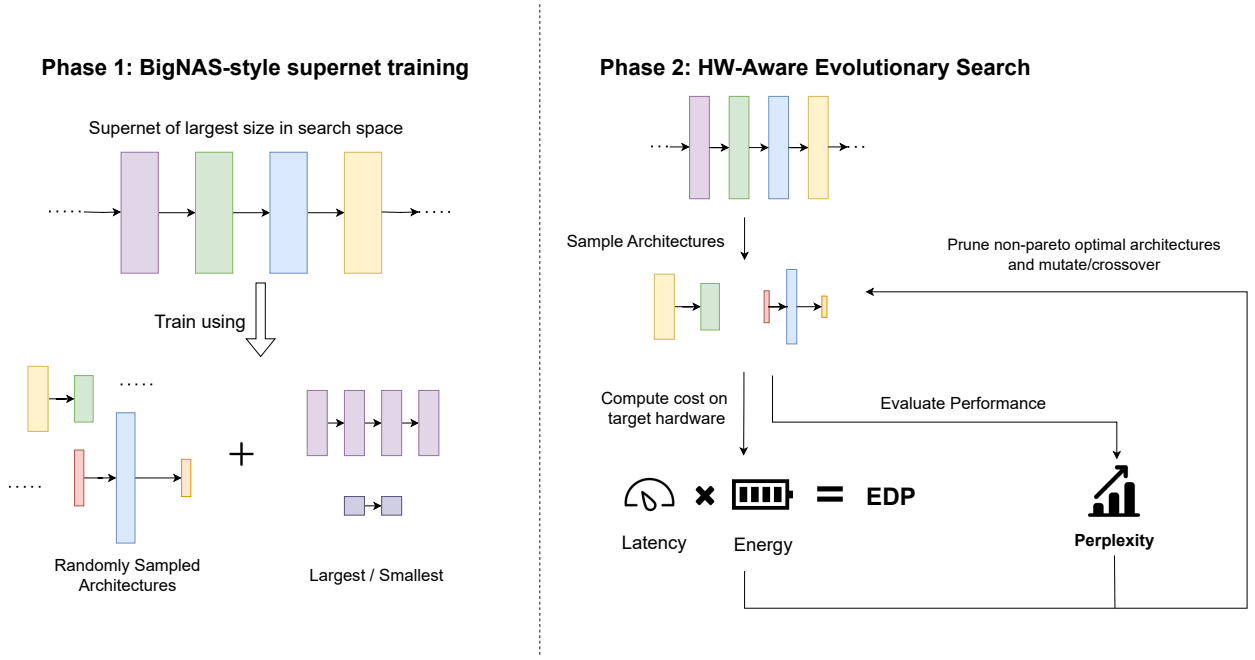

Figure 3.1: NAS Framework from Kim et al. 2023 [19]

Parameter	Range of Values
N	$\{3, 4, 5, 6\}$
h	$\{4, 6, 8, 10, 12\}$
d	$384 - 768$, step size=96
d_{FFN}	$768 - 3072$, step size=128

Table 3.1: NAS Search Space.

For the target hardware, we use an optimized version of Gemmini with dedicated normalization units for running non-linear operations on-chip that was implemented by Hasan Genc and Ruohan Yan. Based on some initial experiments with the Neural Architecture Search framework, we also configure Gemmini with a scratchpad size of 64kB and accumulator size of 256kB to maximize the available accumulator size, as this seemed to lead to better performance overall.

For more details on the experimental methodology, please refer to the survey paper [19].

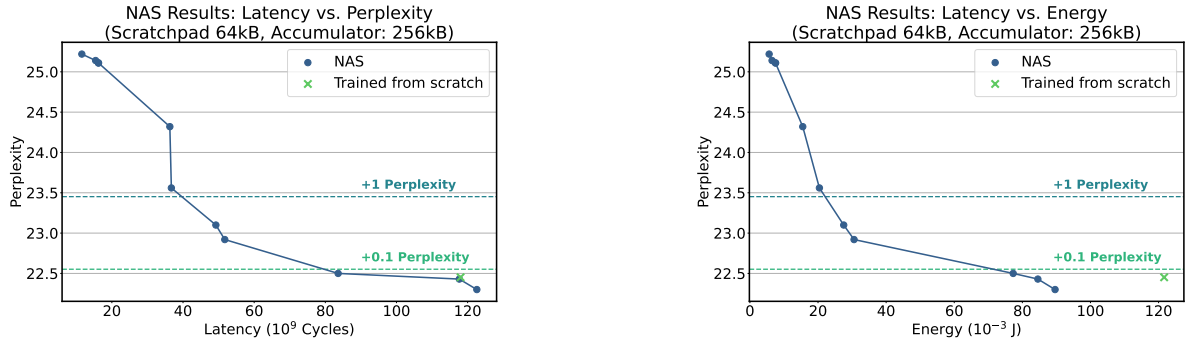

Figure 3.2: (Left) Latency-perplexity and (Right) Energy-perplexity plots of the Transformer architectures found via evolutionary search on our optimal Gemmini hardware configuration. Lower perplexity indicates better performance, and we plot lines to illustrate +0.1 and +1 point perplexity degradation.

3.4 Results

The results from NAS including are shown in the tables below. As can be seen in Fig. 3.2, the NAS framework allows us to obtain multiple Transformer architectures with good hardware cost to perplexity trade-offs. Note that for energy, we use values from Timeloop [23], and for latency, we use RTL simulated values from Ruohan Yan and Hasan Genc. The key findings are summarized below:

- A diverse search space is important, as many ideal architectures have varying number of heads and fully connected layer dimensions between layers. For example, one architecture we found that the number of heads varied between 6 and 12, and the fully connected layer dimensions varied from 768 to 2560, the maximum permitted by our search space.
- We achieved 2.2x EDP reduction with imperceptible perplexity degradation. If we are willing to lerate 1 point of perplexity degradation, these gains can go up to 10.6x on our target hardware.
- The supernet can offer the same perplexity at the same or lower EDP cost compared to the model trained from scratch, and we think this may be due to the regularizing effect of BigNAS.

Chapter 4

Architectural Optimization of Small Transformers

4.1 Overview

Although large language models have shown impressive performance, they are somewhat impractical to run on the edge. This has motivated a whole series of work in designing more efficient architectures such as SqueezeNet [18] or MobileBERT [28]. We focus specifically on the sub-BERT regime (less than 300M parameters). Transformers at this scale have the potential to run quickly on edge and mobile hardware, and if we can make them perform well on specific tasks there will be many applications that open up.

We focus specifically on FFN runtime due to results found from our survey paper [19] that at low sequence lengths, FFN is roughly half of the total runtime for a Transformer block. Furthermore, we have spoken with a large cloud hyperscaler and they have indicated that most of their workloads are the aforementioned short sequence length workloads.

4.2 Experimental Setup

We run our experiments using the T5 model from [26]. T5 is an encoder-decoder foundation language models that is the basis of many modern advances such as the multilingual model mT5 [41], the Byt5 [41] trained on UTF-8 Bytes directly, Long-T5 [15] trained with longer attention context length, and Flan-T5 [8], which was instruction-finetuned. Specifically, we use the T5-Mini variant from [30]. T5-Mini's architecture is presented in Tab. 4.1.

We pretrain the model on the C4 dataset using a batch size of 128 for 524,288 steps using an inverse square-root learning rate schedule. Then, we fine-tune for 262,144 steps with a constant learning rate of 10^{-3} on a proportional mix of all GLUE tasks [36] such that the model sees an equal proportion of examples from each task since some tasks are smaller than others.

Table 4.1: T5-Mini Model Specifications

Parameter	**Value**
Model	T5-Mini
N_L (number of layers)	4/4
d_{ff} (FFN Dimension)	1536
d_{model} (Hidden dimension)	384
d_{kv} (Key-value vector size)	32
N_H (Number of heads)	8
#Params	31M

For benchmarking, we measure the maximum value over three runs on MNLI-mm, one of the tasks from the GLUE benchmark. This was done as we had observed high variance in results, and often one of several runs would have drastically lowered performance compared to the rest of the group. By taking the maximum, we can lower this variance when we want to compare ablations.

4.3 Results

We present our results as a series of ablations based on a specific idea, and in the end present a pareto frontier curve that illustrates the tradeoff between FLOPs and parameter count versus performance on MNLI-mm.

Ablations

Skipping FFN

First we want to establish the contribution of the FFN layer, and so we ran experiments that skip the FFN block completely. The results are described in Tab. 4.2.

Model	**max(MNLI-mm)**	**Difference**	**FLOPs (M)**	**Params (M)**
Baseline	78.2	0.0	1.909	15.351
Skip FFN	75.5	-2.7	0.682	5.907
Skip Encoder FFN	76.1	-2.1	1.296	10.629

Table 4.2: Skip FFN Ablation

In general, it seems that the FFN layer is responsible for 2.7 MNLI-mm points, and we have around 9.4M parameters to try and reinvest for better performance.

Training Longer

Next, we wanted to establish the upper bound for performance that can be achieved if we trained longer. The training settings that we are using is the T5-baseline setting, which was optimized for running many ablations due to the number of ablations that are present in [26]. However, the published T5 models are trained on significantly more data. We wanted to investigate how much more training and data benefits performance as has been studied in Chinchilla [16] and LLaMA [34]. The results are described in Tab. 4.3.

Steps	**max(MNLI-mm)**	**Difference**	**Tokens (B)**	**FLOPs (M)**	**Params (M)**
500k	78.2	0.0	34	1.909	15.351
1M	79.0	0.8	68	1.909	15.351
2M	79.4	1.2	136	1.909	15.351
5M	80.1	1.9	340	1.909	15.351

Table 4.3: Training Longer

As one can see, although there are still some gains to be had, the advantage of training longer quickly diminishes, as going from 500k to 1M steps yielded a 0.8 point increase but going from 1M to 2M only yielded a 0.4 point increase, and 2M to 5M only as 0.5 point increase.

Weight Reparametrization

Next we looked at the idea of structural weight reparametrization, which has been suggested in the Computer Vision area before in works such as [12]. The idea is to design a train time architecture that can be reparametrized into a more efficient inference time architecture. For example, during training you may have a linear neural network that computes $AB\mathbf{x}$ where A and B are suitable matrices, but at inference you may reparametrize them as $C = AB$ and apply just $C\mathbf{x}$ instead. With greater capacity at training time, you may achieve better performance than training using the smaller reparametrized matrix.

In our experiments, we trained the following linear neural architectures for the FFN as described in 4.4. We exclude the ReLU activation in the middle and keep the bias term for now.

As the results in 4.5 indicate, there is a a lot of promise in using bigger weight matrices at train time, using 3 weight matrices was only 1.0 points off the baseline, which has ReLU in between the weight matrices. We may be able to reinvest the parameters saved here into making the model bigger.

Name	Weight 1	Activation	Weight 2	Weight 3
Baseline	1536x384	ReLU	384x1536	-
1W	384x384	-	-	-
1W w/ ReLU	384x384	ReLU	-	-
2W	1536x384	-	384x1536	-
3W	1536x384	ReLU	1536x1536	384x1536

Table 4.4: Reparametrizable Architectures Studied

Model	max(MNLI-mm)	Difference	FLOPs (M)	Params (M)
Baseline	78.2	0.0	1.909	15.351
1W	76.5	-1.7	1.291	10.626
1W w/ ReLU	76.1	-2.1	1.299	10.626
2W	76.7	-1.5	1.291	10.626
3W	77.2	-1.0	1.291	10.626

Table 4.5: FFN Reparametrization Results

Adjusting FFN Expansion with Weight Reparametrization

Expansion Rate	max(MNLI-mm)	Difference	FLOPs (M)	Params (M)
Baseline 4x	77.2	0.0	1.291	10.626
2x	77.3	0.1	1.291	10.626
1.5x	76.7	-0.5	1.291	10.626
1.2x	76.6	-0.6	1.291	10.626

Table 4.6: Adjusting FFN Expansion with Weight Reparametrization

Next, we combine the idea of structural weight reparametrization with reducing the FFN expansion rate from the baseline 4x, in order to see how this changes the performance. We use three linear weight matrices (3W) for all experiments in this subsection.

Our results in Tab. 4.6 indicate that performance is generally correlated with the FFN expansion rate, but that this only matters up to about 2x FFN expansion.

Adjusting Head Dimensions

We also examine the effect of adjusting the number and dimensions of heads to see if further gains can be made here. Our results in Tab. 4.7 indicate that in general you can increase performance by 1-2 points with up to 3M increase in parameter count, but this requires more investigation.

Architecture	**max(MNLI-mm)**	**Difference**	**FLOPs (M)**	**Params (M)**
Baseline	78.2	0.0	1.909	15.351
1.5x # Heads	79.2	1.0	2.246	16.530
2x Head Size	79.2	1.0	2.245	16.530
Both combined	80.0	1.8	2.741	18.273

Table 4.7: Adjusting Head Dimensions

Chapter 5

Conclusion

In this thesis, I discussed two lines of research about optimizing the performance of modern Transformer models. For the co-design direction, we managed to achieve a 2.2× EDP reduction with no perceptible performance loss and a $10.6x$ reduction if one point is tolerable. For optimizing the architecture of smaller models, we were able to reduce the parameter count and inference FLOPs of T5-Mini by about 60% with only a 2.7 point loss on MNLI-mm, a standard natural language inference benchmark. We also perform a number of other ablations and find that one can use structural weight reparametrization to reduce inference FLOPs and parameters by about 30% with only a one point drop on MNLI-mm.

There is plenty of work that could be done in both directions. For the co-design aspect, we should look into better ways to jointly optimize the hardware and Transformer architecture at the same time, perhaps by using a similar method to [37], as this paves the way for an end-to-end flow that does not require two stages. For the architecture aspect, we should look into where we can reinvest the saved parameters. A first step would be to try and scale up the modified Transformers to match the parameter count of the baseline and see if we can exceed the performance of the original model. Experiments on other tasks must also be investigated as it is not known whether how our architectural changes affect performance on other tasks such as open-ended text generation or question-answer for example. We can also look into alternative FFN structures such as those that rely on structured matrices [10] or sparsity [6].

Bibliography

[1] Alon Amid et al. "Chipyard: Integrated design, simulation, and implementation framework for custom socs". In: *IEEE Micro* 40.4 (2020), pp. 10–21.

[2] Iz Beltagy, Matthew E. Peters, and Arman Cohan. *Longformer: The Long-Document Transformer*. 2020. arXiv: 2004.05150 [cs.CL].

[3] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[4] Han Cai, Ligeng Zhu, and Song Han. "Proxylessnas: Direct neural architecture search on target task and hardware". In: *arXiv preprint arXiv:1812.00332* (2018).

[5] Han Cai et al. "Once-for-all: Train one network and specialize it for efficient deployment". In: *arXiv preprint arXiv:1908.09791* (2019).

[6] Rewon Child et al. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509* (2019).

[7] Krzysztof Choromanski et al. *Rethinking Attention with Performers*. 2022. arXiv: 2009.14794 [cs.LG].

[8] Hyung Won Chung et al. *Scaling Instruction-Finetuned Language Models*. 2022. arXiv: 2210.11416 [cs.LG].

[9] Tri Dao et al. *FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness*. 2022. arXiv: 2205.14135 [cs.LG].

[10] Tri Dao et al. "Kaleidoscope: An efficient, learnable representation for all structured linear maps". In: *arXiv preprint arXiv:2012.14966* (2020).

[11] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[12] Xiaohan Ding et al. "Repvgg: Making vgg-style convnets great again". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 13733–13742.

[13] Hasan Genc et al. "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration". In: *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE. 2021, pp. 769–774.

[14] Amir Gholami. *AI and Memory Wall — medium.com*. https://medium.com/riselab/ai-and-memory-wall-2cb4265cb0b8. [Accessed 01-May-2023].

[15] Mandy Guo et al. "LongT5: Efficient Text-To-Text Transformer for Long Sequences". In: *CoRR* abs/2112.07916 (2021). arXiv: 2112.07916. URL: https://arxiv.org/abs/2112.07916.

[16] Jordan Hoffmann et al. "Training compute-optimal large language models". In: *arXiv preprint arXiv:2203.15556* (2022).

[17] Weizhe Hua et al. *Transformer Quality in Linear Time*. 2022. arXiv: 2202.10447 [cs.LG].

[18] Forrest N Iandola et al. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 MB model size". In: *arXiv preprint arXiv:1602.07360* (2016).

[19] Sehoon Kim et al. *Full Stack Optimization of Transformer Inference: a Survey*. 2023. arXiv: 2302.14017 [cs.CL].

[20] Sehoon Kim et al. "I-bert: Integer-only bert quantization". In: *International conference on machine learning*. PMLR. 2021, pp. 5506–5518.

[21] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. *Reformer: The Efficient Transformer*. 2020. arXiv: 2001.04451 [cs.LG].

[22] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[23] Angshuman Parashar et al. "Timeloop: A systematic approach to dnn accelerator evaluation". In: *2019 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE. 2019, pp. 304–315.

[24] Alec Radford et al. "Improving language understanding by generative pre-training". In: (2018).

[25] Alec Radford et al. "Language models are unsupervised multitask learners". In: *OpenAI blog* 1.8 (2019), p. 9.

[26] Colin Raffel et al. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *The Journal of Machine Learning Research* 21.1 (2020), pp. 5485–5551.

[27] Pranav Rajpurkar et al. "Squad: 100,000+ questions for machine comprehension of text". In: *arXiv preprint arXiv:1606.05250* (2016).

[28] Zhiqing Sun et al. "Mobilebert: a compact task-agnostic bert for resource-limited devices". In: *arXiv preprint arXiv:2004.02984* (2020).

[29] Yi Tay et al. "Efficient transformers: A survey". In: *ACM Computing Surveys* 55.6 (2022), pp. 1–28.

[30] Yi Tay et al. "Scale Efficiently: Insights from Pre-training and Fine-tuning Transformers". In: *CoRR* abs/2109.10686 (2021). arXiv: 2109.10686. URL: https://arxiv.org/abs/2109.10686.

[31] Yi Tay et al. *Sparse Sinkhorn Attention*. 2020. arXiv: 2002.11296 [cs.LG].

[32] Yi Tay et al. *Synthesizer: Rethinking Self-Attention in Transformer Models*. 2021. arXiv: 2005.00743 [cs.CL].

[33] *The WikiText Long Term Dependency Language Modeling Dataset — blog.salesforceairesearch.com*. https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset/. [Accessed 08-May-2023].

[34] Hugo Touvron et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[35] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[36] Alex Wang et al. "GLUE: A multi-task benchmark and analysis platform for natural language understanding". In: *arXiv preprint arXiv:1804.07461* (2018).

[37] Hanrui Wang et al. "Hat: Hardware-aware transformers for efficient natural language processing". In: *arXiv preprint arXiv:2005.14187* (2020).

[38] Sinong Wang et al. *Linformer: Self-Attention with Linear Complexity*. 2020. arXiv: 2006.04768 [cs.LG].

[39] Bichen Wu et al. "Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 10734–10742.

[40] Yunyang Xiong et al. *Nyströmformer: A Nyström-Based Algorithm for Approximating Self-Attention*. 2021. arXiv: 2102.03902 [cs.CL].

[41] Linting Xue et al. "mT5: A massively multilingual pre-trained text-to-text transformer". In: *CoRR* abs/2010.11934 (2020). arXiv: 2010.11934. URL: https://arxiv.org/abs/2010.11934.

[42] Jiahui Yu et al. "Bignas: Scaling up neural architecture search with big single-stage models". In: *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*. Springer. 2020, pp. 702–717.

[43] Manzil Zaheer et al. *Big Bird: Transformers for Longer Sequences*. 2021. arXiv: 2007.14062 [cs.LG].