Exploring the Limits of Small Language Models

Nicholas Lee
Kurt Keutzer
Gopala Krishna Anumanchipalli

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-141
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-141.html

May 12, 2023

Acknowledgement

I would like to thank Professor Kurt Keutzer and Dr. Amir Gholami for their continuous
support and advice on this project and my research career. They have consistently been
there to guide me in my research career thus far. I would also like to thank Ja (Thanakul)
Wattanawong for working together with me and assisting me with completing this project.
I also have to thank Karttikeya Mangalam, Sehoon Kim, Sheng Shen, Coleman Hooper, and
Xiuyu Li for their continued advice on navigating and exploring different research directions
during this project. Thanks to Professor Kurt Keutzer and Professor Gopala Anumanchipalli

for their feedback on this thesis. I am grateful to have worked with everyone on this project.

Exploring the Limits of Small Language Models

by Nicholas Z Lee

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)

* * * * * * *

Professor Gopala Krishna Anumanchipalli
Second Reader

05 / 11 / 2023

(Date)

Abstract

Exploring the Limits of Small Language Models

by

Nicholas Z Lee

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Kurt Keutzer, Chair

With the emergence of a plethora of Large Language Models (LLMs) to date, the future of having LLMs run locally at the edge has come closer and closer with every passing day. However, there has not been as much work on smaller language models that can potentially solve tasks where it would be inefficient to run a full LLM at scale. In this paper, we explore Small Language Models (SLMs) and how we can make them more efficient at the edge without sacrificing performance. Pruning or simplifying SLMs can cause a significant degradation of downstream performance. To this end, we investigate weight reparameterization and knowledge distillation as two avenues for these small language models to mitigate these pitfalls. This study investigates the structure of the FFN module in the transformer architecture in order to improve the inference speed of these language models for short sequence length tasks. We also investigate whether we can distill from these LLMs into significantly smaller SLMs in order to take advantage of the plethora of pretrained models available to the public. We find that when simplifying the FFN module, one can use weight reparameterization at training time to help the model converge and improve downstream accuracy. We also find that knowledge distillation may not be a surefire way to improve the downstream model performance as the difference between the model capacities of these LLMs and small language models may be difficult to overcome.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank Professor Kurt Keutzer and Dr. Amir Gholami for their continuous support and advice on this project and my research career. They have consistently been there to guide me in my research career thus far. I would also like to thank Ja (Thanakul) Wattanawong for working together with me and assisting me with completing this project. I also have to thank Karttikeya Mangalam, Sehoon Kim, Sheng Shen, Coleman Hooper, and Xiuyu Li for their continued advice on navigating and exploring different research directions during this project. Thanks to Professor Kurt Keutzer and Professor Gopala Anumanchipalli for their feedback on this thesis. I am grateful to have worked with everyone on this project.

Chapter 1

Introduction

In the last couple of months, we have seen a variety of research into Large Language Models come onto the scene pushing the frontier of what kinds of large language models can be run reliably and efficiently. What began with the release of LLaMa [34] has expanded to include the release of many variants trained on top of LLaMa such as Alpaca [31], Koala [13] and Vicuna [3] as well as other publicly released models such as Pythia [1] and Stability LM [29]. This Cambrian Explosion of different large language models has led to an increased interest in the idea of running LLMs locally, instead of going to the cloud. This could allow for many different applications where data cannot leave the device such as in legal or medical settings or in resource constrained applications such as on edge devices or personal computers.

In this work, we investigate small language models (<1B parameters) and in particular, investigate how we can make these models run more efficiently without sacrificing performance. To this end, we investigate two main avenues of interest:

- Weight Reparameterization ([11],[12],[10]) is a technique where the neural network has a complex structure during pretraining that can be collapsed down to a more efficient structure at inference time. While we pay an increased cost at training time, being able to reduce the footprint of these models at inference time will lead to reduced latency and power costs during the lifetime of the model.

- Knowledge Distillation [16] is a technique where the information from a teacher model is distilled into a smaller model by providing pseudo-labels for the inputs. With the advent of so many new LLMs coming into the fray, being able to take advantage of these models in order to supercharge smaller models at the edge would prove to be useful at allowing SLMs to handle tasks in a cost or resource constrained environment.

We make the following contributions:

- We investigate the extent to which these small language models have been trained and investigate the FFN module, as it takes up most of the latency for short sequence length applications [19]. We find that one can remove the FFN module and cut the

model size and FLOPs by two-thirds, but we also see a significant drop in test time performance.

- We investigate ways to use weight reparameterization to improve the performance of these small architectures without reducing model performance or accuracy at test time. We find that weight reparameterization can mitigate these drawbacks of pruning the FFN module.
- We use KD to test the efficacy of distilling from these large language models. We find that while KD can improve the performance of these models, it may not be straightforward to distill from much larger language models as the model capacity of the student models may not be enough to emulate the teacher.

Chapter 2

Background and Prior Work

2.1 T5

T5 [26] is a Transformer LLM released by Google. Unlike current GPT-style LLMs [2] which are decode-only or BERT-like [9] encoder-only architectures, T5 uses an encoder-decoder style architecture similar to that of the original Transformer [35] architecture. Since the initial release, several variants of T5 have been built on-top of the original architecture. mT5 [41] is a multilingual variant of T5 pretrained on data covering over 100 languages. ByT5 [40] is a token-free model that uses the byte representation of text as the vocabulary. Flan-T5 [6] was one of the first instruction-finetuned LLMs built on-top of T5 that dramatically improved its performance on a variety of different NLP benchmarks.

A similar publication in the direction of this work is [32], which analyzed the effect of both model size and model shape of T5 on both the upstream pretraining loss and the downstream finetuning performance. The upstream task was i.i.d. denoising on the C4 dataset [26] and the downstream tasks were performance on a multitude of different NLP tasks such as GLUE [36] and SQuAD [27]. This paper found that upstream pretraining loss did not directly correlate with downstream performance. It instead found that at different model sizes, the shape of the model played a significant role in the final accuracy of the model after finetuning. While this work was primarily focused on scaling towards larger and larger variants of T5, in this project, we will instead go the opposite direction, and see what can be gained at the smaller end of the spectrum.

2.2 Efficient Transformers

There have been many papers over the years exploring the realm of efficient transformers such as [30, 18, 22, 7, 9]. Many follow the vein of finding more efficient attention mechanisms: [21, 37, 42, 33, 5, 39, 15, 8, 24, 25, 28, 17, 23]. These works find more efficient approximations or implementations of attention so that for long form content, the quadratic complexity of

the attention mechanism can be reduced or linearized to become more efficient. Some like [23] even replace the attention mechanism all together.

While these works mainly target the attention mechanism, the attention mechanism does not dominate the runtime of these large language models at every scale. A recent survey paper [19] found that the latency of these large language models is dominated by the large matrix multiplications of the FFN modules of the transformer architecture for sequence lengths less than 1024. In fact, for sequences of length 128, the FFN module dominates over half of the latency of BERT-Base. For this reason, we investigate the structure of the FFN module in these language models to see if we can improve their performance for these short sequence length scenarios.

2.3 Reparameterization

Reparameterization is a general technique when at training time, the neural network has a more complex architecture that can be collapsed into a simpler and more efficient mechanism at inference time with no loss to performance. The standard example is the scale and bias parameters of a normalization layer, which is commonly absorbed into the subsequent FFN layer. Reparameterization is commonly used in computer vision, e.g. [11], [12], and [10].

While this structure can be reduced with no loss to accuracy, it is still important for this kind of structure to exist during pretraining as it helps the model converge. For example, in [20], the authors found that removing an extra normalization term in between blocks resulted in worse performance overall. The key was that while the back to back normalization had no effect, the difference between the scale and bias terms of the activations that go into the skip connection and the activations that go into the attention module significantly affected downstream performance.

2.4 Distillation

Knowledge Distillation [16] is a method to transfer information and knowledge from a pretrained teacher model to a smaller student model. The standard distillation methodology is to run the input through both the pretrained teacher model as well as the student model. Then the logits of the output of the teacher model are used as pseudo-labels for the input and can be used in conjunction with the ground truth from the dataset. Knowledge Distillation have been used to augment the training language models in the past, such as in [30] and [38]. For a more comprehensive survey of knowledge distillation, see [14].

With the advent of so many LLMs being released to the public, it would be good to study whether we can distill the performance from these very large models down to smaller language models in order to get an easy boost to performance. It should be noted that KD is not always beneficial to the training of neural networks. In [4], the authors launched a study of KD in vision with an emphasis on analyzing the relationship between teacher model

performance and size with student model downstream performance. The authors found that KD can hinder the downstream performance of the student model. In addition, they found that increasing the size and improving the performance of the teacher model did not correlate to improvement of the student model. In essence, they hypothesize that there is a disconnect between the training objective of learning from the teacher and the downstream task itself. Furthermore, it is conjectured that the smaller student models may not have the capacity to emulate and learn from the teacher model. The authors find ways of circumventing this by using smaller teacher models, stopping distillation early, and using early stopped teachers.

Chapter 3

Methodology

Following [32], we utilize the smaller T5 variants in order to run our ablations. The architecture parameters are shown in Table 3.1. We are interested in how we can make the extremely small T5-Tiny and T5-Small models perform better. In particular, we would like to see what the upper limit in performance of these models is, modulo training resources and time.

3.1 FFN

The FFN module in the transformer architecture typically consists of an expansion layer from d_{model} to d_{ff}, an activation function, and then a projection layer from d_{ff} to d_{model}. Taking into account the insights from [19], we focus on ablating the FFN module of T5 to see how we can improve the efficiency of the model. To this end, we see how the model performs under the following adjustments:

1. Skipping the FFN altogether: Since the model is so small, the W_0 component of the

Model	N_L	d_{ff}	d_{model}	d_{kv}	N_H	#Params
Tiny	4/4	1024	256	32	4	16M
Mini	4/4	1536	384	32	8	31M
Small	6/6	2048	512	32	8	60M
Base	12/12	3072	768	64	12	220M

Table 3.1: T5 Model Sizes from [32]. In this case, N_L represents the number of layers in the encoder and decoder parts of the model. For example, 4/4 means that there are 4 encoder and 4 decoder blocks. d_{ff} is the dimension of the feed forward module, d_{model} is the hidden dimension of the model, d_{kv} is the dimension of the attention heads and N_H is the number of attention heads.

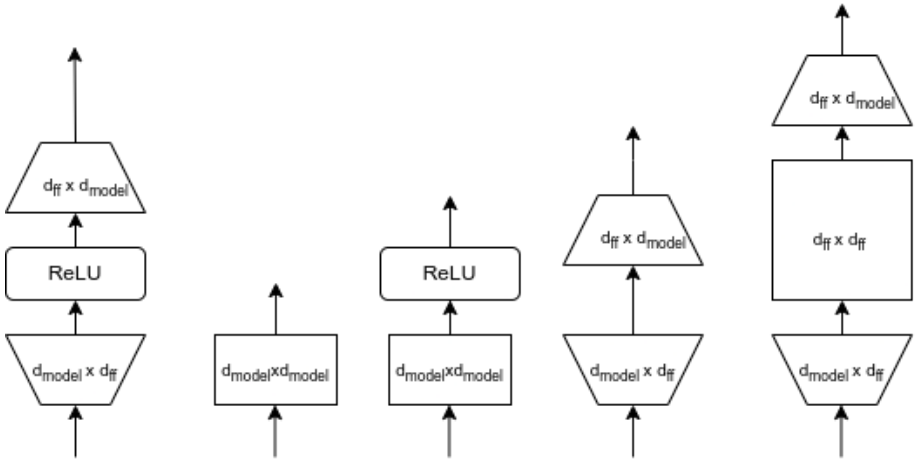

Figure 3.1: Diagram of proposed weight reparameterization methodology. On the very left is the standard FFN module in a transformer (with ReLU as a stand in for the activation function). The second and third and proposed architectures that reduce the FLOPs requirement of the model significantly. The fourth and fifth at inference time will have the same number of parameters and FLOPs as the second, but the weight reparameterization can help stabilize training.

attention mechanism acts like a pseudo-FFN already, just without the skip connection.

2. Adjusting Head Dimensions

3. Weight Reparameterization

With respect to item 3, we would like to see if we can simplify the FFN module for tiny models by removing the expansion and contraction component of the module. However, since this may have adverse effect on the performance of the model, we use weight reparameterization in order to stabilize the training of the model. Figure 3.1 shows the proposed ablations for the FFN module. Each of the 4 on the right would reduce the number of parameters and FLOPs of the FFN layer significantly.

3.2 Knowledge Distillation

We apply knowledge distillation to T5-mini and T5-tiny from T5-Small and T5-Base. Given a temperature T and logits $l(x)$, we can define the smoothed probability distribution

$$p_j^T(x) = \frac{exp(l_j(x)/T)}{\sum_k exp(l_k(x)/T)} \tag{3.1}$$

where $p_j^T(x)$ is the probability of token j given input x. Given the smoothed teacher distribution p_k^t and the smoothed student distribution p_k^s, the total loss would be

$$\mathcal{L} = \alpha \mathcal{L}_{cls} + (1 - \alpha)\mathcal{L}_{KD} \tag{3.2}$$

where

$$\mathcal{L}_{KD} = -T^2 \sum_k p_k^t(x) \log p_k^s(x) \tag{3.3}$$

is the KD loss and $\mathcal{L}_{cls}$ is the standard cross entropy loss.

Chapter 4

Results and Analysis

4.1 Experimental Setup

In the following experiments, we follow the baseline methodology for pretraining and finetuning the models as in [32] and [26]. We train with a batch size of 128 and pretrained for 524,288 Steps with an inverse square root learning rate scheduler on the i.i.d. denoising task on the C4 dataset. During finetuning, we finetune on GLUE for 262,144 steps with a constant 10^{-3} learning rate.

While pretraining and finetuning these models, we found that these smaller models suffer from instability. As seen in Figure 4.1, some of the runs fail to converge during pretraining and subsequently have poor performance during finetuning. We hypothesize that this is due to poor initialization, as restarting training from an intermediate checkpoint did not yield any of this high variance behavior. Interestingly, we did not see this behavior when pretraining T5-Base or T5-Small which suggests that this is an issue that affect only models at this scale. Across many runs, the behavior is that the outlier always performs poorly compared to the base model. In order to mitigate this issue, we did extensive hyperparameter tuning and found that for smaller models, using a linear warmup of 20,000 steps (compared to a constant warmup of 10,000 steps from [26]) manages to reduce the variance across the board.

In the following results, we always report the **maximum** result across 5 runs in order to eliminate the effect that this outlier has on our results. This comes from the fact that we never found an outlier that performed better than the other runs, only worse. We report our results using MNLI-mm (mismatched) from the GLUE test suite.

4.2 Architecture Ablations

In order to establish a baseline on what kind of improvements we can gain, we first experiment with removing the FFN module in Table 4.1. Based on this, we can see that removing the FFN module completely from the model only results in a 2.7 drop in performance, in exchange for a hefty drop in parameter count and FLOPs.

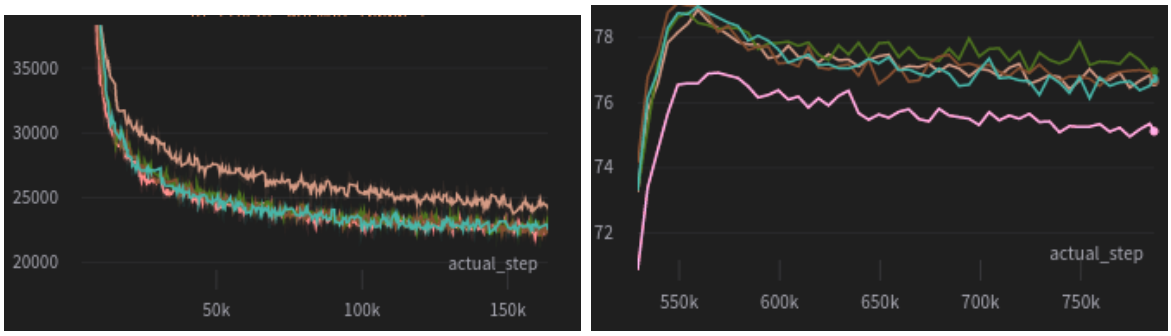

Figure 4.1: On the left is a pretraining loss curve for T5-Mini on the exact same settings but 5 different random seeds. Early on, one of the runs fails to converge and this leads to degraded performance downstream. We mitigate this by treating that run as an outlier and reporting only the maximum value across all 5 runs.

Taking a step further, we also experiment with training for more steps. The results in Table 4.2 show there are diminishing returns to training longer on the model. These results align themselves well with [26] in that it shows that the baseline model is pretrained sub-optimally. We also explored adjusting the head size and head dimensions of the attention module in Table 4.3. While increasing the number of heads and the size of the heads improves performance overall, it also comes with a non-trivial increase in parameter count and FLOPs.

Table 4.4 shows the results of the weight reparameterization experiments on T5-mini. As we can see, replacing the FFN module with just one weight matrix leads to a 33% drop in FLOPs and parameters, but leads to significant degradation in performance. However, by using weight reparameterization with two or three weight matrices that can be re-parameterized at inference time, we can recover some of the performance back. Overall, this shows that we can cut the parameters and FLOPs of this model in exchange for only a 1 point drop in MNLI-mm.

Going further, we experimented on changing the FFN expansion rate of the FFN module for the 3 weight reparameterization scheme. The idea here is that the second weight matrix of size $d_{ff} \times d_{ff}$ can have a significant impact on pretraining, so reducing the expansion rate from $4x$ can help make this method more lightweight for pretraining. As seen in Table 4.5, the FFN expansion can have some effect on the final finetuning performance. Reducing the FFN expansion too much led to degraded performance as after reducing beyond $1.5x - 1.2x$ expansion, the model now performs worse than just having two weight matrices. This shows that having a large expansion factor is still important for pretraining even if the weights will be reparameterized at runtime.

	MNLI-mm	Improvement (+)	FLOPs(M)	Parameters (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 4.1: Performance of T5-Mini after removing the FFN from the whole model and from just the encoder. Surprisingly, we can remove the FFN completely with only a 2.7 drop in performance. This establishes a lower bound for our FFN experiments and guides our plan to recover this accuracy.

Steps	MNLI-mm	Improvement (+)	Training Tokens
Baseline 500k	78.2	0.0	34B
1M	79.0	+0.8	68B
2M	79.4	+1.2	136B
5M	80.1	+1.9	340B

Table 4.2: Training T5-Mini for more and more steps. Similarly to [32] and [26], we find that the baseline model is undertrained.

	MNLI-mm	Improvement (+)	FLOPS (M)	Parameters (M)
Baseline	78.2	0.0	1.909	15.351
1.5x Heads ($N_H = 12$)	79.2	1.0	2.246	16.530
2x Head Size ($d_{kv} = 64$)	79.2	1.0	2.245	16.530
1.5x Heads, 2x Head Size	80.0	1.8	2.741	18.273

Table 4.3: Training T5-Mini with different head dimensions. Increasing the head size or head dimension improves performance significantly, but increases the cost of running the model.

4.3 Knowledge Distillation

Table 4.6 shows the results from the knowledge distillation experiments. As we can see, distillation manages to improve the downstream performance of the T5-Mini and T5-Tiny models. However, we see that distilling from T5-Small rather than T5-Base manages to significantly reduce the variance of the result while also maintaining or even improving on the downstream performance of the model. This aligns well with the results from [4], and shows evidence that distilling from a larger language model can have limited results if the smaller language model does not have the capacity to replicate the larger model's representations. Going forward, it seems like it is important to consider the relative size differences between the teacher and student models when using knowledge distillation.

Weights	MNLI-mm	Improvement (+)	FLOPS (M)	Parameters (M)
Baseline	78.2	0.0	1.909	15.351
1W	76.5	-1.7	1.291	10.626
1W + 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 4.4: Training T5-Mini with different weight reparameterization techniques. The experiments with 2 and 3 weight matrices show that weight reparameterization indeed improves the model's downstream performance.

	MNLI-mm	Improvement (+)	FLOPS (M)	Parameters (M)
Baseline	78.2	0.0	1.909	15.351
4x Expansion, 3W	77.2	-1.0	1.291	10.626
2x Expansion, 3W	77.3	-0.9	1.291	10.626
1.5x Expansion, 3W	76.7	-1.5	1.291	10.626
1.2x Expansion, 3W	76.6	-1.6	1.291	10.626

Table 4.5: Training T5-Mini three weight matrices and different expansion rates. FFN expansion is important during training, but can be collapsed during deployment for efficient inference.

Setup	MNLI-mm	Stdev	Improvement (+)	α	T
Mini (Baseline)	78.2	0.32			
Small→Mini	79.5	**0.56**	**+1.3**	0.1	3
Base→Mini	79.4	2.18	+1.2	0.9	4
Tiny (Baseline)	76.3	0.39			
Small→Tiny	77.7	**0.99**	**+1.4**	0.1	2
Base→Tiny	77.7	3.35	+1.4	0.9	4

Table 4.6: Distillation results for T5-Mini and T5-Small. As you can see, distilling from a smaller model reduced the variance of the downstream performance of the model while maintaining or improving on the performance compared to distilling from the T5-Base model.

Chapter 5

Conclusion

In conclusion, we have investigated very small language models and found some interesting results. The FFN can be removed from the model completely in order to achieve a two-thirds drop in parameter count and FLOPs, while only giving up a 2.7 drop in finetuning accuracy. Reducing the FFN layer to a simple projection layer can reduce the parameter count by one-third and the use of weight reparameterization can reduce the performance derogation from 1.7 to 1.0 points on MNLI-mm. We have also investigated the use of distillation in language models. While KD improves the performance of the model, it is also important to take into account the relative difference between the capacities and models sizes of the teacher and students models in order to achieve good performance. In particular, we found that while distilling from T5-Base model improved performance on T5-Mini and T5-Tiny by 1.2 and 1.4 points respectively, distilling from T5-Small resulted in improved performance by 1.3 and 1.4 points respectively while also significantly reducing the variance of the distillation experiments. These results can help us develop newer and more efficient architectures that can run more efficiently at inference time.

In the future, further work can be done to see what kind of changes can be done in the small model regime. One idea would be to somehow merge the MHA and FFN modules into one. The output layer of the MHA layer is a projection layer that looks very similar to the 1 weight matrix FFN variant, except that that variant includes a skip connection from the input to the MHA layer. It would be interesting to see if we can develop a model without this skip connection, which would improve the memory and computation costs of the model at runtime.

Bibliography

[1] Stella Biderman et al. *Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling*. 2023. arXiv: 2304.01373 [cs.CL].

[2] Tom B. Brown et al. *Language Models are Few-Shot Learners*. 2020. arXiv: 2005.14165 [cs.CL].

[3] Wei-Lin Chiang et al. *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality*. Mar. 2023. URL: https://lmsys.org/blog/2023-03-30-vicuna/.

[4] Jang Hyun Cho and Bharath Hariharan. *On the Efficacy of Knowledge Distillation*. 2019. arXiv: 1910.01348 [cs.LG].

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

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

[7] Kevin Clark et al. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators*. 2020. arXiv: 2003.10555 [cs.CL].

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

[9] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. 2019. arXiv: 1810.04805 [cs.CL].

[10] Xiaohan Ding et al. *ACNet: Strengthening the Kernel Skeletons for Powerful CNN via Asymmetric Convolution Blocks*. 2019. arXiv: 1908.03930 [cs.CV].

[11] Xiaohan Ding et al. *Diverse Branch Block: Building a Convolution as an Inception-like Unit*. 2021. arXiv: 2103.13425 [cs.CV].

[12] Xiaohan Ding et al. *RepVGG: Making VGG-style ConvNets Great Again*. 2021. arXiv: 2101.03697 [cs.CV].

[13] Xinyang Geng et al. *Koala: A Dialogue Model for Academic Research*. Blog post. Apr. 2023. URL: https://bair.berkeley.edu/blog/2023/04/03/koala/ (visited on 04/03/2023).

[14] Jianping Gou et al. "Knowledge Distillation: A Survey". In: *International Journal of Computer Vision* 129.6 (Mar. 2021), pp. 1789–1819. DOI: 10.1007/s11263-021-01453-z. URL: https://doi.org/10.1007%2Fs11263-021-01453-z.

[15] Albert Gu, Karan Goel, and Christopher Ré. *Efficiently Modeling Long Sequences with Structured State Spaces*. 2022. arXiv: 2111.00396 [cs.LG].

[16] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML].

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

[18] Forrest N. Iandola et al. *SqueezeBERT: What can computer vision teach NLP about efficient neural networks?* 2020. arXiv: 2006.11316 [cs.CL].

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

[20] Sehoon Kim et al. *Squeezeformer: An Efficient Transformer for Automatic Speech Recognition*. 2022. arXiv: 2206.00888 [eess.AS].

[21] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. "Reformer: The Efficient Transformer". In: *CoRR* abs/2001.04451 (2020). arXiv: 2001.04451. URL: https://arxiv.org/abs/2001.04451.

[22] Zhenzhong Lan et al. *ALBERT: A Lite BERT for Self-supervised Learning of Language Representations*. 2020. arXiv: 1909.11942 [cs.CL].

[23] James Lee-Thorp et al. *FNet: Mixing Tokens with Fourier Transforms*. 2022. arXiv: 2105.03824 [cs.CL].

[24] Xuezhe Ma et al. *Mega: Moving Average Equipped Gated Attention*. 2023. arXiv: 2209.10655 [cs.LG].

[25] Jason Phang, Yao Zhao, and Peter J. Liu. *Investigating Efficiently Extending Transformers for Long Input Summarization*. 2022. arXiv: 2208.04347 [cs.CL].

[26] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG].

[27] Pranav Rajpurkar et al. *SQuAD: 100,000+ Questions for Machine Comprehension of Text*. 2016. arXiv: 1606.05250 [cs.CL].

[28] Aurko Roy et al. *Efficient Content-Based Sparse Attention with Routing Transformers*. 2020. arXiv: 2003.05997 [cs.LG].

[29] Stability-AI. *Stability-ai/stablelm: Stablelm: Stability ai language models*. URL: https://github.com/Stability-AI/StableLM.

[30] Zhiqing Sun et al. *MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices*. 2020. arXiv: 2004.02984 [cs.CL].

[31] Rohan Taori et al. *Stanford Alpaca: An Instruction-following LLaMA model*. `https://github.com/tatsu-lab/stanford_alpaca`. 2023.

[32] 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`.

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

[34] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. 2023. arXiv: `2302.13971 [cs.CL]`.

[35] Ashish Vaswani et al. "Attention Is All You Need". In: *CoRR* abs/1706.03762 (2017). arXiv: `1706.03762`. URL: `http://arxiv.org/abs/1706.03762`.

[36] Alex Wang et al. *GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding*. 2019. arXiv: `1804.07461 [cs.CL]`.

[37] Sinong Wang et al. "Linformer: Self-Attention with Linear Complexity". In: *CoRR* abs/2006.04768 (2020). arXiv: `2006.04768`. URL: `https://arxiv.org/abs/2006.04768`.

[38] Mengzhou Xia, Zexuan Zhong, and Danqi Chen. *Structured Pruning Learns Compact and Accurate Models*. 2022. arXiv: `2204.00408 [cs.CL]`.

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

[40] Linting Xue et al. *ByT5: Towards a token-free future with pre-trained byte-to-byte models*. 2022. arXiv: `2105.13626 [cs.CL]`.

[41] Linting Xue et al. *mT5: A massively multilingual pre-trained text-to-text transformer*. 2021. arXiv: `2010.11934 [cs.CL]`.

[42] Manzil Zaheer et al. "Big Bird: Transformers for Longer Sequences". In: *CoRR* abs/2007.14062 (2020). arXiv: `2007.14062`. URL: `https://arxiv.org/abs/2007.14062`.